PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2004-258905
(43)Date of publication of application : 16.09.2004

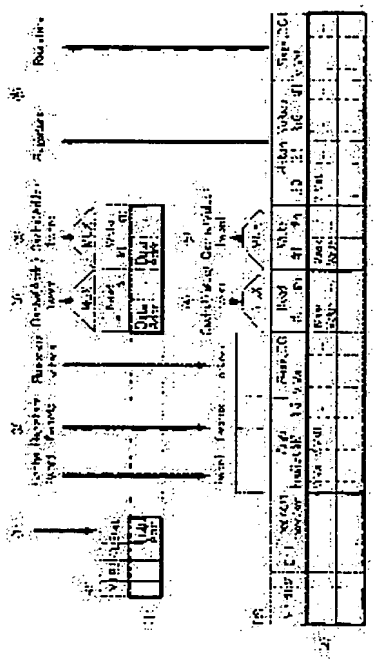
(51)Int.Cl. G06F 9/32

(21)Application number : 2003-047909
(22)Date of filing : 25.02.2003

(71)Applicant : JAPAN SCIENCE & TECHNOLOGY AGENCY
(72)Inventor : NAKAJIMA YASUHIKO

(54) DATA PROCESSING DEVICE AND PROGRAM AND RECORDING MEDIUM FOR RECORDING THIS PROGRAM

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a data processing device for reading a command sequence and/or values from a main storage means to carry out the process of writing the results of operations into the main storage means and capable of increasing the speed of the operations through the reuse of the values even if the command sequence is a loop.
SOLUTION: The start address and end address of a loop to be reused and a main storage address about inputs/outputs are recorded on an RF and information about all registers about the inputs/outputs of the loop and input/output values are recorded on an RB. If the command sequence decoded is a loop and there is a complete match between the contents recorded on the RF and the RB about the loop, the output value recorded on the RB is output as an output about the loop.



LEGAL STATUS

[Date of request for examination] 10.02.2006
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

Japanese Laid-Open Publication

No. 2004-258905 (*Tokukai 2004-258905*)

A. Relevance of the Above-identified Document

The following is a partial English translation of exemplary portions of non-English language information that may be relevant to the issue of patentability of the claims of the present application.

B. Translation of the Relevant Passages of the Document

See the attached English Abstract.

[Claims]

[Claim 12]

A data processing device defined in any one of claims 9-11,

wherein, in each entry of the instruction row storage means, the state of registration and reuse within a predetermined time is stored, E = (the number of steps reduced in the past) × (the number of registration) × (the number of reuse) is worked out, and the second computing means subjects, to precomputation, the entry which has the largest E.

(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2004-258905
(P2004-258905A)
(43)公開日　平成16年9月16日(2004.9.16)

(51)Int.Cl.[7]	F I			テーマコード（参考）
G06F 9/32	G06F	9/32	330A	5B033
	G06F	9/32	330C	

審査請求 未請求　請求項の数 14　OL　（全 31 頁）

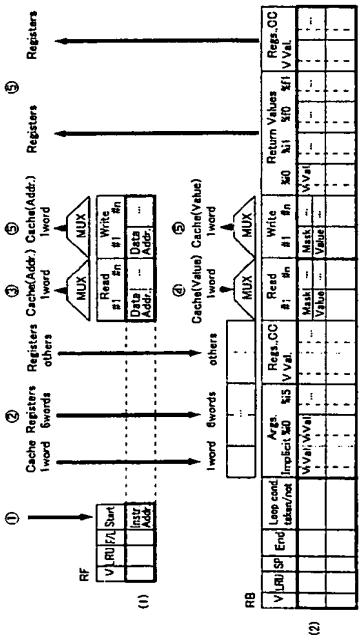
(21)出願番号　特願2003-47909(P2003-47909)
(22)出願日　平成15年2月25日(2003.2.25)

(71)出願人　503360115
独立行政法人　科学技術振興機構
埼玉県川口市本町４丁目１番８号
(74)代理人　100080034
弁理士　原　謙三
(74)代理人　100113701
弁理士　木島　隆一
(74)代理人　100116241
弁理士　金子　一郎
(72)発明者　中島　康彦
京都府京都市左京区吉田神楽岡町５－２８－１０２

Fターム(参考)　5B033 AA03　CA11　CA13

(54)【発明の名称】データ処理装置、データ処理プログラム、およびデータ処理プログラムを記録した記録媒体

(57)【要約】
【課題】主記憶手段から命令列および／または値を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置において、命令列がループである場合にも、値再利用によって演算の高速化を図ることができるデータ処理装置を提供する。
【解決手段】ＲＦに、再利用すべきループの開始アドレス、終了アドレス、および入出力に関する主記憶アドレスを記録し、ＲＢに、該ループの入出力に関する全てのレジスタの情報および入出力値を記録しておく。そして、デコードした命令列がループであった場合、該ループに関して、ＲＦおよびＲＢに登録されている内容と完全に一致した場合、ＲＢに登録されている出力値を、該ループに関する出力として出力する。
【選択図】　図１



【特許請求の範囲】
【請求項１】
主記憶手段から命令列および／または値を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置において、
上記主記憶手段から読み出した命令列をデコードし、該命令列に基づく演算を行う第１の演算手段と、
１つ以上の命令列に関する情報を記憶する命令列記憶手段とを備え、
上記第１の演算手段が、
命令列をデコードした結果、該命令列が後方分岐命令に挟まれたループであるか否かを判断する第１のステップと、
上記第１のステップにおいて、ループであると判断された場合に、該ループに関する情報が上記命令列記憶手段に記憶されているか否かを判断する第２のステップと、
上記第２のステップにおいて、記憶されていないと判断された場合に、該ループを実行する演算を行うとともに、上記命令列記憶手段に対して、演算結果に基づいて該ループに関する入力および出力を記録する第３のステップと、
上記第２のステップにおいて、記憶されていると判断された場合に、処理対象としてのループに対する入力と、上記命令列記憶手段に記憶されているループに対する入力とが一致している場合に、上記命令列記憶手段に記憶されているループに対する出力の値を、処理対象としてのループに対する出力として出力する第４のステップとを行うことを特徴とするデータ処理装置。
【請求項２】
上記第１のステップにおいて、さらに、命令列をデコードした結果、該命令列が関数であるか否かも判断し、
上記第１のステップにおいて、関数であると判断された場合に、該関数に関する情報が上記命令列記憶手段に記憶されているか否かを判断する第５のステップと、
上記第５のステップにおいて、記憶されていないと判断された場合に、該関数を実行する演算を行うとともに、上記命令列記憶手段に対して、演算結果に基づいて該関数に関する入力および出力を記録する第６のステップと、
上記第５のステップにおいて、記憶されていると判断された場合に、処理対象としての関数に対する入力と、上記命令列記憶手段に記憶されている関数に対する入力とが一致している場合に、上記命令列記憶手段に記憶されている関数に対する出力の値を、処理対象としての関数に対する出力として出力する第７のステップとを行うことを特徴とする請求項１記載のデータ処理装置。
【請求項３】
上記第３のステップにおいて、上記命令列記憶手段に対して、主記憶手段における該ループの開始アドレスおよび終了アドレスと、全ての入出力に関する主記憶アドレスとを記録するとともに、
上記第４のステップにおいて、処理対象としてのループの開始アドレス、ならびに、入力に関する主記憶アドレスが、命令列記憶手段に記憶されているものと完全に一致した場合に、上記命令列記憶手段に記憶されているループに対する出力の値を、処理対象としてのループに対する出力として出力することを特徴とする請求項１または２記載のデータ処理装置。
【請求項４】
上記第６のステップにおいて、上記命令列記憶手段に対して、主記憶手段における該関数の先頭アドレスと、入出力に関する主記憶アドレスとを記録するとともに、
上記第７のステップにおいて、処理対象としての関数の先頭アドレス、ならびに、入力に関する主記憶アドレスが、命令列記憶手段に記憶されているものと完全に一致した場合に、上記命令列記憶手段に記憶されている関数に対する出力の値を、処理対象としての関数に対する出力として出力することを特徴とする請求項２記載のデータ処理装置。
【請求項５】

上記主記憶手段から読み出された入力を一時的に格納し、上記第１の演算手段にわたすとともに、上記第１の演算手段からの出力を一時的に格納し、上記主記憶手段にわたすレジスタをさらに備えるとともに、
上記命令列記憶手段に、ループの入力および出力が格納される全てのレジスタの情報が記録されることを特徴とする請求項３記載のデータ処理装置。
【請求項６】
特定の上位の命令区間が、その内部に１つ以上の下位の命令区間を含んでいる場合に、下位の命令区間の実行中に、該下位の命令区間が行った入出力が上位の命令区間の局所変数である場合には、該下位の命令区間の入出力として上記命令列記憶手段に登録し、該下位の命令区間が行った入出力が上位の命令区間の局所変数でない場合には、該下位の命令区間および上位の命令区間の入出力として上記命令列記憶手段に登録することを特徴とする請求項１ないし５のいずれか一項に記載のデータ処理装置。
【請求項７】
上記命令列記憶手段に対して、上位の命令区間および１つ以上の下位の命令区間が登録されている場合に、登録されている命令区間の包含関係の順番をスタック構造で記憶する再利用記憶手段をさらに備えていることを特徴とする請求項６記載のデータ処理装置。
【請求項８】
上記再利用記憶手段に登録可能な命令区間の多重度を超えて、命令区間を検出した場合、その時点で最も上位の命令区間の登録を中止することを特徴とする請求項７記載のデータ処理装置。
【請求項９】
少なくとも１つの第２の演算手段をさらに備え、
上記第２の演算手段が、上記第１の演算手段によって処理が行われている命令列に関して、今後入力が予想される予測入力値に基づいて該命令列の演算を行い、その結果を上記命令列記憶手段に対して登録することを特徴とする請求項１ないし８のいずれか一項に記載のデータ処理装置。
【請求項１０】
上記第１の演算手段に対して、主記憶手段に対して読み出しおよび書き込みが可能なキャッシュメモリが設けられているとともに、上記第２の演算手段に対して、主記憶手段に対して読み出しのみ可能なキャッシュメモリが設けられており、
上記第２の演算手段が、命令列記憶手段に対する入出力記録対象となる主記憶参照には命令列記憶手段に記録されている入出力記録そのものを用い、その他の局所的な参照には、上記第２の演算手段に設けられたローカルメモリを用いるとともに、
上記第１の演算手段が主記憶手段へ書き込みを行う場合には、対応する第２の演算手段のキャッシュラインが無効化されることを特徴とする請求項９記載のデータ処理装置。
【請求項１１】
上記命令列記憶手段において、上記第１の演算手段による書き込み範囲と、上記第２の演算手段による書き込み範囲とを分割するとともに、上記第１の演算手段による書き込み範囲では、ＬＲＵ（ｌｅａｓｔ　ｒｅｃｅｎｔｌｙ　ｕｓｅｄ）によってエントリの入れ替えが行われるようにし、上記第２の演算手段による書き込み範囲では、ＦＩＦＯ（Ｆｉｒｓｔ　Ｉｎ　Ｆｉｒｓｔ　Ｏｕｔ）によってエントリの入れ替えが行われるようにすることを特徴とする請求項９または１０記載のデータ処理装置。
【請求項１２】
命令列記憶手段における各エントリごとに、一定期間における登録および再利用の状況が記録され、Ｅ＝（過去の削減ステップ数）×（登録回数）×（再利用回数）が算出されるとともに、上記第２の演算手段が、上記Ｅが最大となるエントリに関して事前実行を行うことを特徴とする請求項９ないし１１のいずれか一項に記載のデータ処理装置。
【請求項１３】
請求項１ないし１２のいずれか一項に記載のデータ処理装置が備える第１の演算手段が行う処理をコンピュータに実行させることを特徴とするデータ処理プログラム。

【請求項１４】
請求項１ないし１２のいずれか一項に記載のデータ処理装置が備える第１の演算手段が行う処理をコンピュータに実行させることを特徴とするデータ処理プログラムを記録した記録媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は、主記憶手段から命令列および／または値を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置に関するものである。
【０００２】
【従来の技術】
従来、ＣＰＵ（Ｃｅｎｔｒａｌ　Ｐｒｏｃｅｓｓｉｎｇ　Ｕｎｉｔ）を始めとするマイクロプロセッサにおいて、演算速度の高速化技術に関する研究開発が盛んに行われている。高速化技術としては、例えばパイプライン、スーパースケーラ、アウトオブオーダー実行、および、レジスタリネーミングなどが挙げられる。
【０００３】
パイプラインは、命令の実行処理を数段階に分解し、複数の命令を流れ作業的に同時処理を行う技術である。スーパースケーラは、命令の実行回路を２組以上用意し、複数の命令を同時に並行して実行する技術である。アウトオブオーダー実行は、命令の記述順序を無視して、いくつかの連続する命令の中から先に実行可能なものを探して先行処理を行う技術である。レジスタリネーミングは、例えばＣＩＳＣ（Ｃｏｍｐｌｅｘ　Ｉｎｓｔｒｕｃｔｉｏｎ　Ｓｅｔ　Ｃｏｍｐｕｔｅｒ）タイプのプロセッサにおいて、従来のプロセッサにおける命令の互換性を保ちながら、汎用レジスタの数を増やすことによって並行処理が行われる確率を増大させる技術である。
【０００４】
このように、マイクロプロセッサにおける演算速度の高速化を図る際には、命令の実行を並行して行うことが重要となっている。しかしながら、プログラム中には、ある命令の結果に応じて異なる命令が行われるような依存関係、言い換えれば分岐が含まれている場合がほとんどである。このような分岐が含まれている場合、並行処理によって先行して処理を行っていると、分岐の結果によって先行処理した内容が無駄になるという状況が発生することになり、演算速度の高速化の効果が小さくなるという問題がある。
【０００５】
そこで、プログラム中に分岐がある場合に、分岐先を予測することによって先行処理が無駄になる確率を低減し、並行処理の効果を向上させる技術、いわゆる分岐予測に関する研究が数多く行われている。
【０００６】
しかしながら、分岐予測に基づいて投機的先行処理を行う場合には、一般的に次のような問題がある。第１の問題としては、予測の正当性を常に検証する必要があるので、先行命令列の実行時間そのものを削減することはできない、という点である。第２の問題としては、誤った予測に基づく一連の先行演算結果を全て無効化する必要があるので、一度に投機的先行処理できる命令数を多くするには、相応のハードウェアコストを要する、という点である。第３の問題としては、命令間の依存関係が多いほど、多重に投機的先行処理をする必要が生じ、予測の正当性の検証処理、および誤った予測に基づく処理の無効化処理が極めて複雑になる、という点である。
【０００７】
一方、分岐予測とは異なる高速化技術として、値再利用という技術も提案されている。この値再利用とは、プログラムの一部分に関する入力値および出力値を再利用表に登録しておき、同じ箇所を再度実行する際に、入力値が再利用表に登録されているものである場合には、登録されている出力値を出力する、という技術である。この値再利用による効果としては次のようなものが挙げられる。（１）入力値が、再利用表に登録されている入力値

と一致すれば、実行結果を検証する必要がない。（２）入力値および出力値の総数によってのみハードウェアコストが決定され、省略可能な命令列の長さが制約されない。（３）命令間の依存関係の多少は、再利用機構の複雑さに影響を与えない。（４）冗長なロード／ストア命令を削減することができるとともに、これに伴う消費電力の削減も実現される。

【０００８】

後記する非特許文献１には、プログラムにおける関数に関して値再利用を行う技術が示されている。この従来技術では、一般的にロードモジュールがＡＢＩ（Ａｐｐｌｉｃａｔｉｏｎ Ｂｉｎａｒｙ Ｉｎｔｅｒｆａｃｅ）に従って作られることを利用しており、特に、ＳＰＡＲＣ（Ｓｃａｌａｂｌｅ Ｐｒｏｃｅｓｓｏｒ ＡＲＣｈｉｔｅｃｔｕｒｅ）ＡＢＩを利用している。そして、このＡＢＩにおいて関数の入出力を特定することによって値再利用を実現している。すなわち、値再利用のためのコンパイラによる専用命令の埋め込みが不要となっており、既存ロードモジュールへの適用が可能となっている。

【０００９】

また、関数の多重構造を動的に把握することにより、関数内局所レジスタやスタック上の局所変数を値再利用における入出力値から除外するようにしており、これによって効率を向上させている。特に関数については、関数の複雑さに拘わらず、最大６のレジスタ入力、最大４のレジスタ出力、および、局所変数を含まない最小限の主記憶値の登録による再利用および事前実行が可能となっている。この従来技術について以下に詳細に説明する。

【００１０】

まず、単一の関数を対象として、何が入力で何が出力であるかを明らかにし、１レベルの再利用を行うために必要な機構について説明する。プログラムにおいては、一般的に関数は多重構造を形成している。関数Ａ（Ｆｕｎｃｔｉｏｎ－Ａ）が関数Ｂ（Ｆｕｎｃｔｉｏｎ－Ｂ）を呼び出す構造を図１３（ａ）に示す。

【００１１】

帯域変数（Ｇｌｏｂａｌｓ）は、関数Ａの入出力（$Ａ_{in}$／$Ａ_{out}$）および関数Ｂの入出力（$Ｂ_{in}$／$Ｂ_{out}$）になりうるものである。関数Ａの局所変数（Ｌｏｃａｌｓ－Ａ）は、関数Ａの入出力ではないが、ポインタを通じてＢの入出力になりうるものである。また、関数Ａから関数Ｂへの引数（Ａｒｇｓ）は、関数Ｂへの入力となりうるものであり、関数Ｂから関数Ａの返り値（Ｒｅｔ．Ｖａｌ．）は、関数Ｂからの出力となりうるものである。なお、関数Ｂの局所変数（Ｌｏｃａｌｓ－Ｂ）は、関数Ａおよび関数Ｂの入出力には含まれない。

【００１２】

コンテクストに依存せずに関数Ｂを再利用するには、関数Ｂの実行時に、関数Ｂの入出力$Ｂ_{in}$／$Ｂ_{out}$のみを入出力として登録しなければならない。ここで、図１３（ａ）に示すプログラム構造を実行する際の主記憶におけるメモリマップを図１３（ｂ）に示す。このメモリマップにおいて、$Ｂ_{in}$／$Ｂ_{out}$を含まない領域はＬｏｃａｌｓ－Ｂのみとなっている。よって、$Ｂ_{in}$／$Ｂ_{out}$を識別するには、ＧｌｏｂａｌｓとＬｏｃａｌｓ－Ｂとの境界、および、Ｌｏｃａｌｓ－ＢとＬｏｃａｌｓ－Ａとの境界をそれぞれ確定しなければならない。前者については、一般的にＯＳ（Ｏｐｅｒａｔｉｎｇ Ｓｙｓｔｅｍ）が実行時のデータサイズおよびスタックサイズの上限を決めることを利用し、ＯＳが設定する境界（ＬＩＭＩＴ）に基づいてＧｌｏｂａｌｓとＬｏｃａｌｓ－Ｂとの境界を確定することができる。後者については、Ｂが呼び出される直前のスタックポインタの値（ＳＰ ｉｎ Ａ）を用いることによって、Ｌｏｃａｌｓ－ＢとＬｏｃａｌｓ－Ａとの境界を確定することができる。

【００１３】

次に、与えられた主記憶アドレスが、大域変数であるか、または、どの関数の局所変数であるかを識別する方法について説明する。ロードモジュールは、ＳＰＡＲＣ ＡＢＩに規定されている以下の条件を満たすと仮定する。なお、％ｆｐはフレームポインタ、％ｓｐはスタックポインタを意味するものとする。

▲１▼％ｓｐ以上の領域のうち、％ｓｐ＋０～６３はレジスタ退避領域、％ｓｐ＋６８～９１は引数退避領域であり、いずれも関数の入出力ではない。
▲２▼構造体を返す場合の暗黙的引数（Ｉｍｐｌｉｃｉｔ　Ａｒｇ．）は％ｓｐ＋６４～６７に格納される。
▲３▼明示的引数（Ｅｘｐｌｉｃｉｔ　Ａｒｇ．）はレジスタ％ｏ０～５、％ｓｐ＋９２以上の領域に置かれる。
【００１４】
まず、大域変数と局所変数とを区別するために、一般的に、ＯＳが実行時のデータサイズおよびスタックサイズの上限を決めることを利用し、次の事項を仮定する。
▲１▼大域変数はＬＩＭＩＴ未満の領域に置かれる。
▲２▼％ｓｐは、ＬＩＭＩＴ以下になることはなく、ＬＩＭＩＴ～％ｓｐの領域は無効である。
【００１５】
以上の条件を満たしながら、関数Ａが関数Ｂを呼び出す場合の、メモリマップにおける引数およびフレームの概要を図１４に示す。同図を参照しながら、以下にＡの局所変数およびＢの局所変数を区別する方法について説明する。
【００１６】
同図において、（ａ）はＡ実行中の状態を示している。ＬＩＭＩＴ未満の太枠部分に命令（Ｉｎｓｔｒｕｃｔｉｏｎｓ）および大域変数（Ｇｌｏｂａｌ　Ｖａｒｓ．）が格納され、％ｓｐ以上に有効な値が格納されている。％ｓｐ＋６４には、Ｂが構造体を返り値とする場合の暗黙的引数として、構造体の先頭アドレスが格納される。Ｂに対する明示的引数の先頭６ワードはレジスタ％ｏ０～５、第７ワード以降は％ｓｐ＋９２以上に格納される。ベースレジスタを％ｓｐとするオペランド％ｓｐ＋９２が出現した場合、この領域は引数の第７ワードすなわちＢの局所変数である。一方、オペランド％ｓｐ＋９２が出現しない場合、この領域はＡの局所変数である。このように、（ａ）の状態では、オペランドを検証することによってＡの局所変数とＢ局所変数とを区別することができる。
【００１７】
一方、（ｂ）はＢ実行中の状態を示している。引数が入力、返り値が出力、大域変数およびＡの局所変数が入出力となりうる。ただし、Ｂは可変長引数を受け入れる場合があるので、一般に％ｆｐ＋９２以上の領域がＡの局所変数の領域となるかＢの局所変数の領域となるかは判断できない。
【００１８】
局所変数を区別するには、まず、（ａ）の時点において引数の第７ワード以降を検出した関数呼び出しは再利用の対象外とし、第７ワード以降を検出しない関数呼び出しに関して、直前に％ｓｐ＋９２の値を記録しておくようにする。なお、第７ワード以降を使用する関数呼び出しの出現頻度が低いと予想されることから、第７ワード以降を使用する関数を再利用の対象外とする制限による性能低下は軽微なものと考える。
【００１９】
以上の準備により、（ｂ）における主記憶参照アドレスが、予め記録した％ｓｐ＋９２の値以上の場合はＡの局所変数、小さい場合はＢの局所変数であることがわかる。Ｂ実行時には、Ｂの局所変数を除外しながら、大域変数およびＡの局所変数を再利用表へ登録する。
【００２０】
再利用の際は、Ｂの局所変数は入出力から除外されるので、Ｂの局所変数のアドレスが一致している必要がない。このため、いかなるコンテクストであっても、入力さえ一致すれば、再利用することが可能である。ただし、Ｂが参照する大域変数やＡの局所変数については、アドレスおよびデータの両方が再利用表の内容と完全に一致する必要がある。すなわち、Ｂを実行する前に、どのようにして比較すべき主記憶アドレスを網羅するかがポイントになる。
【００２１】

Ｂが参照する大域変数やＡの局所変数のアドレスは、そもそもＢにおいて生成されるアドレス定数や、大域変数／引数を起源とするポインタに基づいているものである。よって、まず引数が完全に一致する再利用表中のエントリを選択した後に、関連する主記憶アドレスをすべて参照して一致比較を行うことにより、Ｂが参照すべき主記憶アドレスを網羅することができる。そして、全ての入力が一致した場合にのみ、登録済の出力（返り値、大域変数、およびＡの局所変数）を再利用することができる。

【００２２】

関数再利用を実現するために、再利用表として、関数管理表（ＲＦ）および入出力記録表（ＲＢ）を設けることにする。１つの関数を再利用するために必要なハードウェア構成を図１５に示す。複数の関数を再利用可能とするには、この構成を複数組用意することになる。

【００２３】

この表において、ＲＦおよびＲＢに保持されるＶは、エントリが有効であるか否かを示すフラグであり、ＬＲＵ（ｌｅａｓｔ　ｒｅｃｅｎｔｌｙ　ｕｓｅｄ）は、エントリ入れ替えのヒントを示している。ＲＦは、上記のＶおよびＬＲＵの他に、関数の先頭アドレス（Ｓｔａｒｔ）、および参照すべき主記憶アドレス（Ｒｅａｄ／Ｗｒｉｔｅ）を保持する。ＲＢは、上記のＶおよびＬＲＵの他に、関数呼び出し直前の％ｓｐ（ＳＰ）、引数（Ａｒｇｓ．）（Ｖ：有効エントリ、Ｖａｌ．：値）、主記憶値（Ｍａｓｋ：Ｒｅａｄ／Ｗｒｉｔｅアドレスの有効バイト、Ｖａｌｕｅ：値）、および、返り値（Ｒｅｔｕｒｎ　Ｖａｌｕｅｓ）（Ｖ：有効エントリ、Ｖａｌ．：値）を保持する。

【００２４】

返り値は、％ｉ０～１（リーフ関数では％ｏ０～１に読み替える）または％ｆ０～１に格納され、％ｆ２～３を使用する返り値（拡張倍精度浮動小数点数）は対象プログラムには存在しないものと仮定する。ＲｅａｄアドレスはＲＦが一括管理し、ＭａｓｋおよびＶａｌｕｅはＲＢが管理することにより、Ｒｅａｄアドレスの内容とＲＢの複数エントリをＣＡＭ（ｃｏｎｔｅｎｔ－ａｄｄｒｅｓｓａｂｌｅ　ｍｅｍｏｒｙ）により一度に比較する構成を可能としている。

【００２５】

単一の関数を再利用するには、まず、関数実行時に、局所変数を除外しながら、引数、返り値、大域変数および上位関数の局所変数に関する入出力情報を再利用表に登録していく。ここで、読み出しが先行した引数レジスタは関数の入出力として、また、返り値レジスタへの書き込みは関数の出力として登録する。その他のレジスタ参照は登録する必要がない。主記憶参照も同様に、読み出しが先行したアドレスについては入力、書き込みは出力として登録する。

【００２６】

関数から復帰するまでに次の関数を呼び出した場合、または、登録すべき入出力が再利用表の容量を超える、引数の第７ワードを検出する、途中でシステムコールや割り込みが発生する、などの擾乱が発生しなかった場合、復帰命令を実行した時点で、登録中の入出力表エントリを有効にする。

【００２７】

以降、図１５を参照しながら説明すると、関数を呼び出す前に、▲１▼関数先頭アドレスを検索し、▲２▼引数が完全に一致するエントリを選択し、▲３▼関連する主記憶アドレスすなわち少なくとも１つのＭａｓｋが有効であるＲｅａｄアドレスをすべて参照して、▲４▼一致比較を行う。全ての入力が一致した場合に、▲５▼登録済の出力（返り値、大域変数、およびＡの局所変数）を書き戻すことによって、関数の実行を省略することができる。

【００２８】

【非特許文献１】

情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，ＨＰＳ５，ｐｐ．１－１２，Ｓｅｐ．（２００２），“関数値再利用および並列事前実行による高速化技

術”（中島康彦、緒方勝也、正西申悟、五島正裕、森眞一郎、北村俊明、富田眞治）（発行日２００２年９月１５日）

【００２９】

【発明が解決しようとする課題】

上記従来の技術は、関数のみを再利用の対象としたものとなっている。よって、コンパイラが関数をインライン展開したり、関数内に処理量の多いループが存在したりする場合には、演算の高速化をほとんど期待することができないことになる。

【００３０】

本発明は上記の問題点を解決するためになされたもので、その目的は、主記憶手段から命令列および／または値を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置において、命令列がループである場合にも、値再利用によって演算の高速化を図ることができるデータ処理装置を提供することにある。

【００３１】

【課題を解決するための手段】

上記の課題を解決するために、本発明に係るデータ処理装置は、主記憶手段から命令列および／または値を読み出し、演算処理を行った結果を主記憶手段に書き込む処理を行うデータ処理装置において、上記主記憶手段から読み出した命令列をデコードし、該命令列に基づく演算を行う第１の演算手段と、１つ以上の命令列に関する情報を記憶する命令列記憶手段とを備え、上記第１の演算手段が、命令列をデコードした結果、該命令列が後方分岐命令に挟まれたループであるか否かを判断する第１のステップと、上記第１のステップにおいて、ループであると判断された場合に、該ループに関する情報が上記命令列記憶手段に記憶されているか否かを判断する第２のステップと、上記第２のステップにおいて、記憶されていないと判断された場合に、該ループを実行する演算を行うとともに、上記命令列記憶手段に対して、演算結果に基づいて該ループに関する入力および出力を記録する第３のステップと、上記第２のステップにおいて、記憶されていると判断された場合に、処理対象としてのループに対する入力と、上記命令列記憶手段に記憶されているループに対する入力とが一致している場合に、上記命令列記憶手段に記憶されているループに対する出力の値を、処理対象としてのループに対する出力として出力する第４のステップとを行うことを特徴としている。

【００３２】

上記の構成では、主記憶から読み出された命令列が第１の演算手段によって演算され、演算結果が主記憶に書き込まれるようになっている。ここで、第１の演算手段は、命令列をデコードした際に、まず該命令列がループであるか否かを判定する。そして、ループである場合には、そのループが初めて実行される場合には、そのループに基づく演算を行うとともに、命令列記憶手段に入力および出力を記録する処理が行われる。その後、同じループが出現した場合に、命令列記憶手段に記憶されている入力と同じであれば、第１の演算手段がそのループに関する演算処理を行うことなしに、命令列記憶手段に記憶されている出力をそのまま出力するようになっている。

【００３３】

したがって、あるループに関する命令列が複数回行われる場合には、２回目以降では、第１の演算手段による演算処理を行うことなく、命令列記憶手段に記憶されている出力を読み出すという処理のみで対応することが可能となる。これにより、処理時間の大幅な短縮を実現することができる。

【００３４】

また、本発明に係るデータ処理装置は、上記の構成において、上記第１のステップにおいて、さらに、命令列をデコードした結果、該命令列が関数であるか否かも判断し、上記第１のステップにおいて、関数であると判断された場合に、該関数に関する情報が上記命令列記憶手段に記憶されているか否かを判断する第５のステップと、上記第５のステップにおいて、記憶されていないと判断された場合に、該関数を実行する演算を行うとともに、上記命令列記憶手段に対して、演算結果に基づいて該関数に関する入力および出力を記録

する第６のステップと、上記第５のステップにおいて、記憶されていると判断された場合に、処理対象としての関数に対する入力と、上記命令列記憶手段に記憶されている関数に対する入力とが一致している場合に、上記命令列記憶手段に記憶されている関数に対する出力の値を、処理対象としての関数に対する出力として出力する第７のステップとを行う構成としてもよい。

【００３５】

上記の構成によれば、命令列が関数である場合にも、上記のループと同様に値再利用を実現することができる。すなわち、ある関数に関する命令列が複数回行われる場合には、２回目以降では、第１の演算手段による演算処理を行うことなく、命令列記憶手段に記憶されている出力を読み出すという処理のみで対応することが可能となる。これにより、処理時間の大幅な短縮を実現することができる。

【００３６】

また、本発明に係るデータ処理装置は、上記の構成において、上記第３のステップにおいて、上記命令列記憶手段に対して、主記憶手段における該ループの開始アドレスおよび終了アドレスと、全ての入出力に関する主記憶アドレスとを記録するとともに、上記第４のステップにおいて、処理対象としてのループの開始アドレス、ならびに、入力に関する主記憶アドレスが、命令列記憶手段に記憶されているものと完全に一致した場合に、上記命令列記憶手段に記憶されているループに対する出力の値を、処理対象としてのループに対する出力として出力する構成としてもよい。

【００３７】

上記の構成によれば、命令列記憶手段には、ループの開始アドレスおよび終了アドレスと、全ての入出力に関する主記憶アドレスとが記録されていることになる。よって、これらの情報を用いることにより、的確にループを特定することが可能となり、ループに関する再利用を実現することが可能となる。

【００３８】

また、本発明に係るデータ処理装置は、上記の構成において、上記第６のステップにおいて、上記命令列記憶手段に対して、主記憶手段における該関数の先頭アドレスと、入出力に関する主記憶アドレスとを記録するとともに、上記第７のステップにおいて、処理対象としての関数の先頭アドレス、ならびに、入力に関する主記憶アドレスが、命令列記憶手段に記憶されているものと完全に一致した場合に、上記命令列記憶手段に記憶されている関数に対する出力の値を、処理対象としての関数に対する出力として出力する構成としてもよい。

【００３９】

上記の構成によれば、命令列記憶手段には、関数の先頭アドレスと、入出力に関する主記憶アドレスとが記録されていることになる。よって、これらの情報を用いることにより、的確に関数を特定することが可能となり、関数に関する再利用を実現することが可能となる。

【００４０】

また、本発明に係るデータ処理装置は、上記の構成において、上記主記憶手段から読み出された入力を一時的に格納し、上記第１の演算手段にわたすとともに、上記第１の演算手段からの出力を一時的に格納し、上記主記憶手段にわたすレジスタをさらに備えるとともに、上記命令列記憶手段に、ループの入力および出力が格納される全てのレジスタの情報が記録される構成としてもよい。

【００４１】

上記の構成によれば、第１の演算手段は、レジスタの値に基づいて、再利用が可能か否かの判断を行うことができるので、この判断を的確かつ迅速に行うことが可能となる。

【００４２】

また、本発明に係るデータ処理装置は、上記の構成において、特定の上位の命令区間が、その内部に１つ以上の下位の命令区間を含んでいる場合に、下位の命令区間の実行中に、該下位の命令区間が行った入出力が上位の命令区間の局所変数である場合には、該下位の

命令区間の入出力として上記命令列記憶手段に登録し、該下位の命令区間が行った入出力が上位の命令区間の局所変数でない場合には、該下位の命令区間および上位の命令区間の入出力として上記命令列記憶手段に登録する構成としてもよい。

【００４３】

上記の構成によれば、ある命令区間の内部に、さらに他の命令区間が含まれているような多重入出力構造である場合にも、命令列記憶手段に対して、的確に再利用のための情報を記録することが可能となる。したがって、例えば命令列記憶手段に上位の命令区間に関する情報が登録されている場合に、該命令区間が出現した場合に、その入力を参照することによって、下位の命令区間を考慮することなく、的確に該命令区間の再利用を実現することが可能となる。

【００４４】

また、本発明に係るデータ処理装置は、上記の構成において、上記命令列記憶手段に対して、上位の命令区間および１つ以上の下位の命令区間が登録されている場合に、登録されている命令区間の包含関係の順番をスタック構造で記憶する再利用記憶手段をさらに備えている構成としてもよい。

【００４５】

上記の構成によれば、命令列記憶手段に多重入出力構造の命令区間が複数登録されていても、再利用記憶手段を参照することによって、各命令区間の包含関係を認識することが可能となる。

【００４６】

また、本発明に係るデータ処理装置は、上記の構成において、上記再利用記憶手段に登録可能な命令区間の多重度を超えて、命令区間を検出した場合、その時点で最も上位の命令区間の登録を中止する構成としてもよい。

【００４７】

上記の構成によれば、多重入出力構造を登録中に、その構造に含まれる新たな命令区間を検出し、かつ、再利用記憶手段に登録可能な多重度を超えてしまう場合に、最も上位の命令区間に関する登録が中止されることになる。ここで、例えば最も下位の命令区間に関する登録が中止されてしまうと、それまでに登録されていた多重入出力構造は全て使えない状態になってしまうが、上記のようにすれば、再利用可能な多重入出力構造の登録を最低限確保することが可能となる。

【００４８】

また、本発明に係るデータ処理装置は、上記の構成において、少なくとも１つの第２の演算手段をさらに備え、上記第２の演算手段が、上記第１の演算手段によって処理が行われている命令列に関して、今後入力が予想される予測入力値に基づいて該命令列の演算を行い、その結果を上記命令列記憶手段に対して登録する構成としてもよい。

【００４９】

上記の構成によれば、第２の演算手段によって、その時点で第１の演算手段によって処理が行われている命令列に関して、予測入力値に基づく演算が行われ、その結果が命令列記憶手段に記憶されることになる。よって、次に、同じ命令列が出現し、予測入力値と同じ入力が行われた場合には、命令列記憶手段に記憶されている値を再利用することが可能となる。例えば、入力値が単調に変化するような命令列の場合には、予測入力値が的中する可能性が高いので、上記の構成による効果は高くなる。

【００５０】

また、本発明に係るデータ処理装置は、上記の構成において、上記第１の演算手段に対して、主記憶手段に対して読み出しおよび書き込みが可能なキャッシュメモリが設けられているとともに、上記第２の演算手段に対して、主記憶手段に対して読み出しのみ可能なキャッシュメモリが設けられており、上記第２の演算手段が、命令列記憶手段に対する入出力記録対象となる主記憶参照には命令列記憶手段に記録されている入出力記録そのものを用い、その他の局所的な参照には、上記第２の演算手段に設けられたローカルメモリを用いるとともに、上記第１の演算手段が主記憶手段へ書き込みを行う場合には、対応する第

２の演算手段のキャッシュラインが無効化される構成としてもよい。
【００５１】
上記の構成によれば、第２の演算手段は、主記憶手段に対しては読み出しのみを行い、書き込みはできないようになっている。そして、第２の演算手段は、命令列記憶手段に対する入出力記録対象となる主記憶参照には命令列記憶手段に記録されている入出力記録そのものを用い、その他の局所的な参照には、上記第２の演算手段に設けられたローカルメモリを用いるようになっている。これにより、第１の演算手段と、第２の演算手段とで、主記憶手段に対して異なる値を書き込むことによる不具合を防止することができる。
【００５２】
また、第１の演算手段が主記憶手段へ書き込みを行う場合には、対応する第２の演算手段のキャッシュラインが無効化されるので、第１の演算手段によって主記憶手段が書き換えられた場合にも、第２の演算手段が、新たに主記憶手段から値を読み出すことによって、主記憶一貫性を保つことができる。
【００５３】
また、本発明に係るデータ処理装置は、上記の構成において、上記命令列記憶手段において、上記第１の演算手段による書き込み範囲と、上記第２の演算手段による書き込み範囲とを分割するとともに、上記第１の演算手段による書き込み範囲では、ＬＲＵ（ｌｅａｓｔ　ｒｅｃｅｎｔｌｙ　ｕｓｅｄ）によってエントリの入れ替えが行われるようにし、上記第２の演算手段による書き込み範囲では、ＦＩＦＯ（Ｆｉｒｓｔ　Ｉｎ　Ｆｉｒｓｔ　Ｏｕｔ）によってエントリの入れ替えが行われるようにする構成としてもよい。
【００５４】
命令区間は、大きく分けると、第１の演算手段のみでも再利用の効果があるものと、配列を扱うループのように第１の演算手段のみでは効果がないものとに分けられると考えられる。前者であれば、ＬＲＵによる入れ替え、後者であれば、ＦＩＦＯによる入れ替えが有効である。
【００５５】
これに対して、上記の構成によれば、第１の演算手段による書き込み範囲と、第２の演算手段による書き込み範囲とを分割するとともに、エントリの入れ替えは、第１の演算手段による書き込み範囲ではＬＲＵ、第２の演算手段による書き込み範囲ではＦＩＦＯとしているので、より効率的に再利用を行うことが可能となる。
【００５６】
また、本発明に係るデータ処理装置は、上記の構成において、命令列記憶手段における各エントリごとに、一定期間における登録および再利用の状況が記録され、Ｅ＝（過去の削減ステップ数）×（登録回数）×（再利用回数）が算出されるとともに、上記第２の演算手段が、上記Ｅが最大となるエントリに関して事前実行を行う構成としてもよい。
【００５７】
上記の構成によれば、第１の演算手段による登録頻度が高く、かつ、第２の演算手段が登録したエントリの再利用頻度も高い命令列を継続して第２の演算手段の実行対象とすることができる。よって、第２の演算手段による事前実行の効果を最大限高めることが可能となる。
【００５８】
【発明の実施の形態】
本発明の実施の一形態について図１ないし図１２に基づいて説明すれば、以下のとおりである。
【００５９】
（データ処理装置の構成）
本実施形態に係るデータ処理装置の概略構成を図５に示す。同図に示すように、該データ処理装置は、ＭＳＰ（Ｍａｉｎ　Ｓｔｒｅａｍ　Ｐｒｏｃｅｓｓｏｒ）１Ａ、ＳＳＰ（Ｓｈａｄｏｗ　Ｓｔｒｅａｍ　Ｐｒｏｃｅｓｓｏｒ）１Ｂ、再利用表としてのＲＦ／ＲＢ（命令列記憶手段）２、および主記憶（主記憶手段）を備えた構成となっており、主記憶３

に記憶されているプログラムデータなどを読み出して各種演算処理を行い、演算結果を主記憶３に書き込む処理を行うものである。なお、同図に示す構成では、ＳＳＰ１Ｂを１つ備えた構成となっているが、２つ以上備えた構成となっていてもよい。また、同図に示す構成では、ＳＳＰ１Ｂを備えた構成となっているが、ＳＳＰ１Ｂを備えていない構成としてもかまわない。ＳＳＰ１Ｂを備えた場合の作用・効果については、後述する。

【００６０】

ＲＦ／ＲＢ２は、プログラムにおける関数およびループを再利用するためのデータを格納するメモリ手段である。このＲＦ／ＲＢ２の詳細については後述する。

【００６１】

主記憶３は、ＭＳＰ１ＡおよびＳＳＰ１Ｂの作業領域としてのメモリであり、例えばＲＡＭ（Ｒａｎｄｏｍ　Ａｃｃｅｓｓ　Ｍｅｍｏｒｙ）などによって構成されるものである。例えばハードディスクなどの外部記憶手段からプログラムやデータなどが主記憶３に読み出され、ＭＳＰ１ＡおよびＳＳＰ１Ｂは、主記憶３に読み出されたデータに基づいて演算を行うことになる。

【００６２】

ＭＳＰ１Ａは、ＲＷ（再利用記憶手段）４Ａ、演算器（第１の演算手段）５Ａ、レジスタ６Ａ、およびＣａｃｈｅ７Ａを備えた構成となっている。また、ＳＳＰ１Ｂは、同様に、ＲＷ（再利用記憶手段）４Ｂ、演算器（第２の演算手段）５Ｂ、レジスタ６Ｂ、およびＣａｃｈｅ／Ｌｏｃａｌ７Ｂを備えた構成となっている。

【００６３】

ＲＷ４Ａ・４Ｂは、再利用ウィンドウであり、現在実行中かつ登録中であるＲＦおよびＲＢ（後述する）の各エントリをリング構造のスタックとして保持するものである。このＲＷ４Ａ・４Ｂは、実際のハードウェア構造としては、ＲＦ／ＲＢ２における特定のエントリをアクティブにする制御線の集合によって構成される。

【００６４】

演算器５Ａ・５Ｂは、レジスタ６Ａ・６Ｂに保持されているデータに基づいて演算処理を行うものであり、ＡＬＵ（ａｒｉｔｈｍｅｔｉｃ　ａｎｄ　ｌｏｇｉｃａｌ　ｕｎｉｔ）と呼ばれるものである。レジスタ６Ａ・６Ｂは、演算器５Ａ・５Ｂによって演算を行うためのデータを保持する記憶手段である。なお、本実施形態では、演算器５Ａ・５Ｂ、およびレジスタ６Ａ・６Ｂは、ＳＰＡＲＣアーキテクチャに準じたものとする。Ｃａｃｈｅ７Ａ・７Ｂは、主記憶３と、ＭＳＰ１ＡおよびＳＳＰ１Ｂとの間でのキャッシュメモリとして機能するものである。なお、ＳＳＰ１Ｂでは、Ｃａｃｈｅ７Ｂには、局所メモリとしてのＬｏｃａｌ７Ｂが含まれているものとする。

【００６５】

（関数およびループの概念的説明）

ここで、上記データ処理装置の詳細な説明に入る前に、関数とループとの類似性について、図２（ａ）および図２（ｂ）を参照しながら説明しておく。同図（ａ）は、関数が呼び出されて実行される様子を概念的に示しており、同図（ｂ）は、ループ処理が実行される様子を概念的に示している。関数に含まれる命令は、Ｃａｌｌ命令の分岐先（Ｆｕｎｃ．ｓｔａｒｔ）からＲｅｔｕｒｎ命令（Ｆｕｎｃ．ｅｎｄ）（厳密にはディレイスロットを含む）までである。同様に、ループでは、後方分岐命令の分岐先（Ｌｏｏｐ　ｓｔａｒｔ）から、同じ後方分岐命令（Ｌｏｏｐ　ｅｎｄ）までである。したがって、このＬｏｏｐ　ｓｔａｒｔからＬｏｏｐ　ｅｎｄの間の入出力を登録しておくことによって、関数と同様にループを再利用することができる。ただし、関数では局所変数の登録を除外することができる一方、ループでは除外することができない。これは、ＡＢＩでは、ループ内の局所変数を特定することができないからである。したがって、ループを再利用するためには、図１５に示した従来の関数再利用表に対して、新たに項目を拡張する必要があることになる。

【００６６】

（ＲＦ／ＲＢの構成）

図１は、本実施形態におけるＲＦ／ＲＢ２によって実現される再利用表を示している。同図に示すように、ＲＦは、エントリが有効であるか否かを示すＶ、エントリ入れ替えのヒントを示すＬＲＵ、関数の先頭アドレスを示すＳｔａｒｔ、参照すべき主記憶アドレスを示すＲｅａｄ／Ｗｒｉｔｅに加えて、および、関数とループとを区別するＦ／Ｌを保持している。

【００６７】

また、ＲＢは、エントリが有効であるか否かを示すＶ、エントリ入れ替えのヒントを示すＬＲＵ、関数またはループを呼び出す際の直前のスタックポイント％ｓｐを示すＳＰ、引数（Ａｒｇｓ．）（Ｖ：有効エントリ、Ｖａｌ．：値）、主記憶値（Ｍａｓｋ：Ｒｅａｄ／Ｗｒｉｔｅアドレスの有効バイト、Ｖａｌｕｅ：値）、および、返り値（Ｒｅｔｕｒｎ Ｖａｌｕｅｓ）（Ｖ：有効エントリ、Ｖａｌ．：値）に加えて、ループの終了アドレス（Ｅｎｄ）、ループ終了時の分岐方向を示すｔａｋｅｎ／ｎｏｔ、引数や返り値以外のレジスタおよび条件コード（Ｒｅｇｓ．，ＣＣ）を保持している。

【００６８】

上記のＲＦおよびＲＢにおける各項目についてより詳細に説明する。上記Ｖは、上記のようにエントリが有効であるか否かを示すものであるが、具体的には、未登録時には「０」、登録中である場合には「２」、登録済である場合には「１」の値が格納されるようになっている。例えば、ＲＦまたはＲＢを確保する際に、未登録エントリ（Ｖ＝０）があれば、これを使用し、未登録エントリがなければ、登録済エントリ（Ｖ＝１）の中からＬＲＵが最小のものを選択して上書きすることになる。登録中エントリ（Ｖ＝２）は使用中であるので上書きすることはできない。

【００６９】

上記ＬＲＵは、一定時間間隔で右へシフトされていくシフトレジスタの中の「１」の個数を示したものである。ＲＦの場合、このシフトレジスタは、該当エントリに関して、再利用のための登録を行ったか、もしくは再利用を試みた場合に、左端に「１」が書き込まれるようになっている。したがって、該当エントリが頻繁に使用されれば、ＬＲＵは大きな値となり、一定期間使用されなければ、ＬＲＵの値は０となる。一方、ＲＢの場合、シフトレジスタには、該当エントリが再利用された場合に「１」が書き込まれるようになっている。したがって、該当エントリが頻繁に使用されれば、ＬＲＵは大きな値となり、一定期間使用されなければ、ＬＲＵの値は０となる。

【００７０】

上記ＲＢにおける主記憶値のＭａｓｋについて説明する。一般に、アドレスとデータとを１バイトずつ管理することにすれば管理が可能であるが、実際には、４バイト単位でデータを管理する方がキャッシュ参照を高速に行うことができる。そこで、ＲＦでは、主記憶アドレスを４の倍数で記憶するようになっている。一方、管理単位を４バイトとする場合、１バイト分だけをロードすることに対応できるようにするために、４バイトのうちでどのバイトが有効であるかを示す必要がある。すなわち、Ｍａｓｋは、４バイトのうちでどのバイトが有効であるかを示す４ビットのデータとなっている。例えば、Ｃ００１番地から１バイト分をロードした結果、値がＥ８であった場合、ＲＦには、アドレスＣ０００が登録され、ＲＢのＭａｓｋに「０１００」、Ｖａｌｕｅに「００Ｅ８０００」が登録されることになる。

【００７１】

上記の引数や返り値以外のレジスタおよび条件コード（Ｒｅｇｓ．，ＣＣ）について説明する。本実施形態では、ＳＰＡＲＣアーキテクチャレジスタのうち、汎用レジスタ％ｇ０－７、％ｏ０－７、％１０－７、％ｉ０－７、浮動小数点レジスタ％ｆ０－３１、条件コードレジスタＩＣＣ、浮動小数点条件コードレジスタＦＣＣを用いるようになっている（詳細は後述する）。このうち、リーフ関数の入力は汎用レジスタ％ｏ０－５、出力は汎用レジスタ％ｏ０－１、また、非リーフ関数の入力は汎用レジスタ％ｉ０－５、出力は汎用レジスタ％ｉ０－１、になり、入力は、ａｒｇ［０－５］、出力は、ｒｔｉ［０－１］に登録される。ＳＰＡＲＣ－ＡＢＩの規定では、これら以外のレジスタは関数の入出力には

ならないので、関数に関してはＲＢにおける引数（Ａｒｇｓ．）の項で十分である。
【００７２】
一方、ＳＰＡＲＣ－ＡＢＩの規定では、ループの入出力に関しては、用いられるレジスタの種類を特定することはできないので、ループの入出力を特定するには、全ての種類のレジスタに関してＲＢに登録する必要がある。よって、ＲＢにおけるＲｅｇｓ．，ＣＣには、％ｇ０－７、％ｏ０－７、％ｌ０－７、％ｉ０－７、％ｆ０－３１、ＩＣＣ、ＦＣＣが登録されるようになっている。
【００７３】
以上のように、ＲＦ／ＲＢ２において、ＲｅａｄアドレスはＲＦが一括管理し、ＭａｓｋおよびＶａｌｕｅはＲＢが管理している。これにより、Ｒｅａｄアドレスの内容とＲＢの複数エントリをＣＡＭ（ｃｏｎｔｅｎｔ－ａｄｄｒｅｓｓａｂｌｅ ｍｅｍｏｒｙ）によって一度に比較する構成を可能としている。このことについて、以下により詳しく説明する。
【００７４】
一般的に、アドレスが与えられると、そのアドレスに格納された値を参照することができるメモリは、ＲＡＭと呼ばれるメモリである。一方、上記のＣＡＭとは、連想メモリと呼ばれるメモリであり、検索すべき内容が与えられると、そのエントリに対応する信号がＯＮとなるように動作するようになっている。通常は、ＣＡＭはＲＡＭとセットにして用いられる。
【００７５】
ここで、ＣＡＭとＲＡＭとの連携動作について、具体例を挙げて説明する。ＣＡＭに、「５，５，５，５，５」、「１，３，１，１，１」、「１，３，３，５，２」、「６，６，６，６，６」というデータ列がエントリとして登録されており、ＲＡＭに、ＣＡＭにおける各データ列に対応して、「５，５」、「１，１」、「１，２」、「６，６」というデータが登録されているとする。ここで、検索すべきデータ列として、「１，３，３，５，２」をＣＡＭに入力すると、一致するエントリがＯＮとなり、ＲＡＭに登録されている該当するデータ「１，２」が出力されることになる。この具体例と同様の構成および動作によって、上記ＲＢが実現されることになる。
【００７６】
（再利用処理の概略）
次に、関数およびループのそれぞれの場合について、再利用処理の概略について説明する。
【００７７】
まず、関数の場合について説明する。関数から復帰するまでに次の関数を呼び出した場合、または、登録すべき入出力が再利用表の容量を超える、引数の第７ワードを検出する、途中でシステムコールや割り込みが発生する、などの擾乱が発生しなかった場合、復帰命令を実行した時点で、登録中の入出力表エントリを有効にする。
【００７８】
以降、図１を参照しながら説明すると、関数を呼び出す前に、▲１▼ＲＦに登録されている関数の先頭アドレスに、該当関数の先頭アドレスと一致するものがあるかを検索する。一致するものがある場合には、▲２▼ＲＢに登録されている該当関数に関するエントリにおける引数が、呼び出す関数の引数と完全に一致するエントリを選択する。そして、▲３▼関連する主記憶アドレスすなわち少なくとも１つのＭａｓｋが有効であるＲｅａｄアドレスをＲＦからすべて参照して、▲４▼ＲＢに登録されている内容と一致比較を行う。全ての入力が一致した場合に、▲５▼ＲＢに登録済の出力（返り値、大域変数、およびＡの局所変数）を主記憶に書き戻すことによって、関数の実行を省略する、すなわち関数の再利用を実現することができる。
【００７９】
次に、ループの場合について説明する。ループが完了する以前に関数から復帰したり、前記した擾乱が発生したりするなど、ループの入出力登録が中止されなければ、登録中のル

ープに対応する後方分岐命令を検出した時点で、登録中の入出力表エントリを有効にし、そのループの登録を完了する。

【００８０】

さらに、後方分岐命令が成立する場合は、次のループが再利用可能かどうかを判断する。すなわち、図１を参照しながら説明すると、後方分岐する前に、▲１▼ＲＦに登録されているループの先頭アドレスに、該当ループの先頭アドレスと一致するものがあるかを検索する。一致するものがある場合には、▲２▼ＲＢに登録されている該当ループに関するレジスタ入力値が、呼び出すループのレジスタ入力値と完全に一致するエントリを選択する。そして、▲３▼関連する主記憶アドレスをＲＦから全て参照して、▲４▼ＲＢに登録されている内容と一致比較を行う。全ての入力が一致した場合に、▲５▼ＲＢに登録済の出力（レジスタおよび主記憶出力値）を主記憶に書き戻すことによってループの実行を省略する、すなわちループの再利用を実現することができる。

【００８１】

再利用した場合、ＲＢに登録されている分岐方向に基づいて、さらに次のループに関して同様の処理を繰り返す。一方、次のループが再利用不可能であれば、次のループを通常に実行し、ＲＦおよびＲＢへの登録を開始する。

【００８２】

（ループを含む多重再利用）

１レベルで上記のような再利用機構を用いた場合、図１３（ａ）に示した例で言えば、リーフ関数としての関数Ｂや、関数Ｂの内部にあるループＣなどをそれぞれ再利用することが可能となる。これに対して、ある関数を一度実行しただけで、その関数の内部に含まれる関数やループを含む全ての命令区間が再利用可能となるように登録を行う仕組みが多重再利用である。例えば上記の例で言えば、多重再利用によれば、関数Ａを一度実行しただけで、入れ子関係にあるＡ，Ｂ，Ｃの全ての命令区間が再利用可能となる。以下に、多重再利用を実現する上で必要とされる機能拡張について説明する。

【００８３】

図８に、一例として、関数Ａおよび関数Ｄの概念的な構造を示す。同図に示す例では、関数Ａの内部にループＢが存在しており、ループＢの内部にループＣが存在しており、ループＣにおいて関数Ｄが呼び出されるようになっている。そして、関数Ｄの内部にループＥが存在しており、ループＥの内部にループＦが存在している。

【００８４】

図３は、図８に示す関数Ａ，Ｄおよびループ Ｂ，Ｃ，Ｅ，Ｆの入れ子構造において、内側の構造のレジスタ入出力（太枠セル領域）が、外側の構造のレジスタ入出力となる影響範囲（矢印）について示している。例えば、ループＦの内部において入力として参照された％ｉ０～５は、ループＥおよび関数Ｄに対する入力でもあり、さらに、関数Ｄを呼び出したループＣおよびループＢに対する入力（ただし％ｏ０～５に読み替える）でもある。一方、関数Ａにとって％ｏ０～５は局所変数に相当するので、％ｉ０～５（％ｏ０～５）は、関数Ａに対してのレジスタ入力とはならない。すなわち、％ｉ０～５（％ｏ０～５）の影響範囲はループＢまでとなる。別の見方をすれば、関数Ｄの内部で％ｉ０～５が参照された場合には、ループＢが直接的に％ｏ０～５を参照しなくても、％ｏ０～５をループＢの入力値として登録する必要がある。ループＦ内部において出力された％ｉ０～１についても同様である。

【００８５】

浮動小数点レジスタはレジスタウィンドウに含まれないので、出力された％ｆ０～１は、関数Ａを含む全階層の出力となる。一方、その他のレジスタ入出力は、関数を超えて影響がおよぶことはない。すなわち、ループＦ内部における入出力、すなわち、レジスタ入力としての％ｉ６～７、％ｇ，ｌ，ｏ、％ｆ０～３１、％ｉｃｃ、％ｆｃｃ、およびレジスタ出力としての％Ｉ２～７、％ｇ，ｌ，ｏ、％ｆ２～３１、％ｉｃｃ、％ｆｃｃの影響範囲はループＥまでとなる。主記憶に対する入出力については、前述した、関数呼び出し直前の％ｓｐ（ＳＰ）と比較する方法を入れ子の全階層に対して適用することにより、影響

範囲を特定することができる。

【００８６】

以上のことから、多重再利用を実現するには、前述したＲＦおよびＲＢを関数やループの入れ子構造と関連づける機構が必要である。図４に示すように、再利用ウィンドウ（ＲＷ）を装備することによって、現在実行中かつ登録中であるＲＦおよびＲＢの各エントリ（図中ではＡ、Ｂ、Ｃと示す）をスタック構造として保持する。関数やループの実行中は、ＲＷに登録されている全てのエントリについて、これまでに述べた方法に基づいて、レジスタおよび主記憶参照を登録していく。

【００８７】

この際に、あるエントリに関して、（１）登録可能項目数の超過、（２）引数の第７ワードの検出、（３）システムコールの検出、によって再利用不可能であると判断した場合には、ＲＷを用いて、そのエントリに対応するＲＢおよび上位のＲＢを特定し、登録を中止することができる。

【００８８】

なお、ＲＷの深さは有限であるものの、一度に登録可能な多重度を超えて関数やループを検出した場合には、外側の命令区間から順次登録を中止し、より内側の命令区間を登録対象に加えることによって、入れ子関係の動的変化に追随することができる。また、実行および登録中（例えばＡ）に、再利用可能な命令区間（例えばＤ）に遭遇した場合には、登録済の入出力をそのまま登録中エントリに追加することによって、ＲＷの深さを超えるＡの多重再利用も可能となる。

【００８９】

（処理の流れ）

次に、命令がデコードされた場合の具体的な処理の流れについて説明する。以下では、命令がデコードされた結果、関数呼び出し命令である場合、関数復帰命令である場合、後方分岐成立の場合、後方分岐不成立の場合、およびその他の命令の場合について、それぞれ処理の流れを説明する。

【００９０】

（関数呼び出し命令である場合）

命令がデコードされた結果、関数呼び出し命令である場合の処理を図９に示すフローチャートを参照しながら以下に説明する。まずステップ１（以降、Ｓ１のように称する）において、引数の第７ワードを検出したか否かが判定される。Ｓ１においてＹＥＳ、すなわち、引数の第７ワードを検出したと判定された場合には、ＲＷに登録されている登録中ＲＢエントリを全て無効化し、Ｓ６に移行して、プログラムカウンタを関数の先頭へ進め、処理を終了する。

【００９１】

一方、Ｓ１においてＮＯ、すなわち、引数の第７ワードを検出していないと判定された場合には、該関数呼び出しおよび入力値がＲＦおよびＲＢに登録されているか否かを検索する（Ｓ２）。Ｓ２においてＹＥＳ、すなわち、該関数呼び出しおよび入力値がＲＦおよびＲＢに登録されていると判定された場合には、後述するＳ７のステップに移行する。

【００９２】

Ｓ２においてＮＯ、すなわち、該関数呼び出しおよび入力値がＲＦおよびＲＢに登録されていないと判定された場合、該関数のためのＲＦエントリおよびＲＢエントリを確保しようと試み、▲１▼既存のＲＦエントリがあるか、▲２▼登録作業中につき追い出すことのできないＲＦエントリ以外に、使用可能なＲＦエントリがあるか、または▲３▼登録作業中につき追い出すことができないＲＢエントリ以外に、使用可能なＲＢエントリがあるかを判定する（Ｓ３）。

【００９３】

Ｓ３においてＮＯ、すなわち、使用可能なＲＦ・ＲＢエントリがないと判定された場合には、登録を開始せず、ＲＷに登録されているＲＢを全て無効化し（Ｓ５）、ＲＷを空にする。一方、Ｓ３においてＹＥＳ、すなわち、使用可能なＲＦ・ＲＢエントリがあると判定

された場合には、該関数のためのＲＦエントリおよびＲＢエントリを確保し、ＲＷに登録する（Ｓ４）。ここで、ＲＷに登録した際に、ＲＷに登録されているＲＷエントリが溢れた際には、最も古いＲＷエントリを削除し、対応するＲＢを無効化する。Ｓ３またはＳ４が行われた後に、プログラムカウンタを関数の先頭へ進め（Ｓ６）、処理を終了する。

【００９４】

一方、Ｓ２においてＹＥＳ、すなわち、該関数呼び出しおよび入力値がＲＦおよびＲＢに登録されていると判定された場合、該関数は再利用可能であることになる。すなわち、ＲＢから出力値を求めるとともに、レジスタおよび主記憶にこの出力値を書き込む（Ｓ７）。そして、登録中の関数／ループがＲＷに登録されているか否かを判定し（Ｓ８）、登録されている場合には、再利用を行った関数のＲＢエントリの内容のうち必要なものをＲＷに登録されているエントリに追加する（Ｓ９）。ここで、ＲＷのＴＯＰから順に登録し、途中でＲＢがあふれた場合には、以降、ＲＷのＢＯＴＴＯＭまでに対するＲＢを無効化し、ＲＷから削除する。その後、プログラムカウンタを次の命令へ進め（Ｓ１０）、処理を終了する。

【００９５】

（関数復帰命令である場合）

命令がデコードされた結果、関数復帰命令である場合の処理を図１０に示すフローチャートを参照しながら以下に説明する。Ｓ１１において、ＲＷのＴＯＰから順にたどり、関数に対応するＲＦ／ＲＢが検出されるまでに、ループに関するＲＢが検出されるか否かが判定される（Ｓ１２）。ここでループに関するＲＢが検出されると（Ｓ１２においてＹＥＳ）、該当ＲＢを全て無効化するとともに、ＲＷから削除する（Ｓ１３）。

【００９６】

一方、ＲＷ探索中に、該関数に対応するＲＦ／ＲＢを検出したか否かが判定される（Ｓ１４）。ここで、該関数に対応するＲＦ／ＲＢが検出されると（Ｓ１４においてＹＥＳ）、該当ＲＢエントリを有効化するとともに、ＲＷから削除する（Ｓ１５）。

【００９７】

その後、復帰命令を実行し（Ｓ１６）、処理を終了する。

【００９８】

（後方分岐成立である場合）

命令がデコードされた結果、後方分岐成立である場合の処理を図１１に示すフローチャートを参照しながら以下に説明する。まず、ＲＷのＴＯＰから順にたどり、関数に対応するＲＢを検出するか否かが判定される（Ｓ２１）。Ｓ２１においてＹＥＳ、すなわち、関数に対応するＲＢを検出した場合には、後述するＳ２４のステップに移行する。

【００９９】

一方、Ｓ２１においてＮＯ、すなわち、関数に対応するＲＢを検出しない場合には、次に、該後方分岐命令自身のアドレスとＲＢ中のループ終了アドレスとが一致するか否かが判定される（Ｓ２２）。Ｓ２２においてＮＯ、すなわち、該後方分岐命令自身のアドレスとＲＢ中のループ終了アドレスとが一致しないと判定されると、後述するＳ２４のステップに移行する。

【０１００】

Ｓ２２においてＹＥＳ、すなわち、該後方分岐命令自身のアドレスとＲＢ中のループ終了アドレスとが一致すると判定された場合、ＲＷのＴＯＰから該ＲＢの手前までのＲＢを全て無効化し（Ｓ２３）、ＲＷから削除する。また、該ＲＢエントリを有効化し、かつｔａｋｅｎ＝１とし、ＲＷから削除する。

【０１０１】

次に、Ｓ２４において、次ループの先頭アドレスおよび入力値がＲＦおよびＲＢに登録されているか否かが判定される。Ｓ２４においてＹＥＳ、すなわち、次ループの先頭アドレスおよび入力値がＲＦおよびＲＢに登録されている場合には、後述するＳ３０のステップに移行する。

【０１０２】

一方、Ｓ２４においてＮＯ、すなわち、次ループの先頭アドレスおよび入力値がＲＦおよびＲＢに登録されていない場合には、次ループのためのＲＦエントリおよびＲＢエントリを確保しようと試み、▲１▼既存のＲＦエントリがあるか、▲２▼登録作業中につき追い出すことができないＲＦエントリ以外に、使用可能なＲＦエントリがあるか、または▲３▼登録作業中につき追い出すことができないＲＢエントリ以外に、使用可能なＲＢエントリがあるかが判定される（Ｓ２５）。

【０１０３】

Ｓ２５においてＮＯ、すなわち、使用可能なＲＦ・ＲＢエントリがないと判定された場合には、登録を開始せずに、ＲＷに登録されているＲＢを全て無効化し（Ｓ２６）、ＲＷを空にする。その後、Ｓ２９において、プログラムカウンタを条件分岐先へ進め、処理を終了する。

【０１０４】

一方、Ｓ２５においてＹＥＳ、すなわち、使用可能なＲＦ・ＲＢエントリがあると判定された場合には、その使用可能なＲＦ・ＲＢエントリを確保し、確保したＲＦ・ＲＢをＲＷに登録する（Ｓ２７）。また、ＲＢにループ終了アドレス（後方分岐命令自身のアドレス）を登録する。ここで、ＲＷへの登録を行った際にＲＷが溢れた場合には、最も古いＲＷエントリを削除し（Ｓ２８）、それに対応するＲＢを無効化する。その後、Ｓ２９において、プログラムカウンタを条件分岐先へ進め、処理を終了する。

【０１０５】

一方、前記したＳ２４においてＹＥＳとなった場合、次ループは再利用可能であることになるので、ＲＢから出力値を求め、この値をレジスタおよび主記憶に書き込む（Ｓ３０）。ここで、登録中の関数／ループがＲＷに登録されているか否かが判定され（Ｓ３１）、登録されている場合、再利用を行ったループのＲＢエントリの内容のうち必要なものをＲＷに登録されているエントリに追加する（Ｓ３２）。このとき、ＲＷのＴＯＰから順に登録し、途中でＲＢが溢れた場合、以降、ＲＷのＢＯＴＴＯＭまでに対するＲＢを無効化し、ＲＷから削除する。

【０１０６】

その後、プログラムカウンタは、次ループ先頭ではなく、該ＲＢ中のｔａｋｅｎの値に応じて、ｔａｋｅｎ＝１の場合は自命令、ｔａｋｅｎ＝０の場合は、ＲＢ中に記憶しておいたループ終了アドレスの次へ進める。その後、処理を終了する。

【０１０７】

（後方分岐不成立である場合）

命令がデコードされた結果、後方分岐不成立である場合の処理を図１２に示すフローチャートを参照しながら以下に説明する。まず、ＲＷのＴＯＰから順に検索し（Ｓ４１）、関数に対応するＲＢを検出したか否かが判定される（Ｓ４２）。Ｓ４２においてＹＥＳ、すなわち、関数に対応するＲＢを検出したと判定された場合、Ｓ４６においてプログラムカウンタを次命令に進め、処理を終了する。

【０１０８】

Ｓ４２においてＮＯ、すなわち、関数に対応するＲＢを検出していないと判定された場合、該後方分岐命令自身のアドレスとＲＢ中のループ終了アドレスが一致するか否かが判定される（Ｓ４３）。Ｓ４３においてＮＯ、すなわち、該後方分岐命令に対応するＲＦ／ＲＢを検出していないと判定された場合、Ｓ４６においてプログラムカウンタを次命令に進め、処理を終了する。

【０１０９】

一方、Ｓ４３においてＹＥＳ、すなわち、該後方分岐命令に対応するＲＦ／ＲＢを検出したと判定された場合、ＲＷのＴＯＰから該ＲＢの手前までのＲＢを全て無効化し（Ｓ４４）、ＲＷから削除する。また、該ＲＢエントリを有効化し、かつｔａｋｅｎ＝０とし、ＲＷから削除する（Ｓ４５）。その後、Ｓ４６においてプログラムカウンタを次命令に進め、処理を終了する。

【０１１０】

（その他の命令である場合）
次に、命令がデコードされた結果、上記以外のその他の命令である場合について説明する。その他の命令である場合、レジスタＲ／Ｗ、主記憶Ｒ／Ｗが実行される。その際に、ＲＷが空でなければ、以下の手順によってレジスタＲ／Ｗ、主記憶Ｒ／ＷをＲＷに登録されているＲＢに対して登録する。以下では、（１）汎用レジスタＲＥＡＤの場合、（２）汎用レジスタＷＲＩＴＥの場合、（３）浮動小数点レジスタＲＥＡＤの場合、（４）浮動小数点レジスタＷＲＩＴＥの場合、（５）条件コードレジスタＩＣＣ－ＲＥＡＤの場合、（６）条件コードレジスタＩＣＣ－ＷＲＩＴＥの場合、（７）浮動小数点条件コードレジスタＦＣＣ－ＲＥＡＤの場合、（８）浮動小数点条件コードレジスタＦＣＣ－ＷＲＩＴＥの場合、（９）主記憶ＲＥＡＤの場合、（１０）主記憶ＷＲＩＴＥの場合についてそれぞれ説明する。

【０１１１】
（１）汎用レジスタＲＥＡＤの場合
まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして、（１－１）該ＲＢがリーフ関数かつ％ｏ０－６の場合、または該ＲＢが非リーフ関数かつ％ｉ０－６の場合、ａｒｇ［０－５］．Ｖ＝０であれば、ａｒｇ［０－５］．Ｖ＝１に変更し、ａｒｇ［０－５］．Ｖａｌに読み出しデータを記録する。その後、さらにＲＷをたどり、該ＲＢが関数の場合、処理を終了する。一方、該ＲＢが関数ではない（ループである）場合、ａｒｇ［０－５］．Ｖ＝０であれば、ａｒｇ［０－５］．Ｖ＝１に変更し、ａｒｇ［０－５］．Ｖａｌに読み出しデータを記録し、処理を終了する。

【０１１２】
一方、（１－２）該ＲＢがループの場合、（ａ）％ｇ０－７でｇｒｒ［０－７］．Ｖ＝０であれば、ｇｒｒ［０－７］．Ｖ＝１に変更し、ｇｒｒ［０－７］．Ｖａｌに読み出しデータを記録し、処理を終了する。（ｂ）％ｏ０－７でａｒｇ［０－７］．Ｖ＝０であれば、ａｒｇ［０－７］．Ｖ＝１に変更し、ａｒｇ［０－７］．Ｖａｌに読み出しデータを記録し、処理を終了する。（ｃ）％１０－７でｌｒｒ［０－７］．Ｖ＝０であれば、ｌｒｒ［０－７］．Ｖ＝１に変更し、ｌｒｒ［０－７］．Ｖａｌに読み出しデータを記録し、処理を終了する。（ｄ）％ｉ０－７でｉｒｒ［０－７］．Ｖ＝０であれば、ｉｒｒ［０－７］．Ｖ＝１に変更し、ｉｒｒ［０－７］．Ｖａｌに読み出しデータを記録し、次のＲＷエントリに進む。

【０１１３】
（２）汎用レジスタＷＲＩＴＥの場合
まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（２－１）該ＲＢがリーフ関数かつ％ｏ０－５の場合、または該ＲＢが非リーフ関数かつ％ｉ０－５の場合、ａｒｇ［０－５］．Ｖ＝０であれば、以降の読み出しは入力ではないことを示すために、ａｒｇ［０－５］．Ｖ＝２に変更する。さらに、％ｏ０－１／％ｉ０－１について、ｒｔｉ［０－１］．Ｖ＝１に変更し、ｒｔｉ［０－１］．Ｖａｌに書き込みデータを記録する。その後、さらにＲＷをたどり、該ＲＢが関数の場合、処理を終了する。一方、該ＲＢが関数ではない（ループである）場合、ａｒｇ［０－１］．Ｖ＝０であれば、以降の読み出しは入力ではないことを示すために、ａｒｇ［０－１］．Ｖ＝２に変更し、ｒｔｉ［０－１］．Ｖ＝１に変更し、ｒｔｉ［０－１］．Ｖａｌに書き込みデータを記録し、処理を終了する。

【０１１４】
一方、（２－２）該ＲＢがループの場合、（ａ）％ｇ０－７でｇｒｒ［０－７］．Ｖ＝０であれば、ｇｒｒ［０－７］．Ｖ＝２に変更し、ｇｒｒ［０－７］．Ｖａｌに書き込みデータを記録し、処理を終了する。（ｂ）％ｏ０－７でａｒｇ［０－７］．Ｖ＝０であれば、ａｒｇ［０－７］．Ｖ＝２に変更し、ａｒｇ［０－７］．Ｖａｌに書き込みデータを記録し、処理を終了する。（ｃ）％１０－７でｌｒｒ［０－７］．Ｖ＝０であれば、ｌｒｒ［０－７］．Ｖ＝２に変更し、ｌｒｒ［０－７］．Ｖａｌに書き込みデータを記録し、処理を終了する。（ｄ）％ｉ０－７でｉｒｒ［０－７］．Ｖ＝０であれば、ｉｒｒ［０－７

]．Ｖ＝２に変更し、ｉｒｒ［０－７］．Ｖａｌに書き込みデータを記録し、次のＲＷエントリに進む。

【０１１５】

（３）浮動小数点レジスタＲＥＡＤの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（３－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（３－２）該ＲＢがループの場合、ｆｒｒ［０－３１］．Ｖ＝０であれば、ｆｒｒ［０－３１］．Ｖ＝１に変更し、ｆｒｒ［０－３１］．Ｖａｌに読み出しデータを記録し、処理を終了する。

【０１１６】

（４）浮動小数点レジスタＷＲＩＴＥの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（４－１）該ＲＢが関数かつ％ｆ０－１の場合、ｒｔｆ［０－１］．Ｖ＝１に変更し、ｒｔｆ［０－１］．Ｖａｌに書き込みデータを記録する。さらにＲＷをたどり、ｆｒｒ［０－１］．Ｖ＝０であれば、以降の読み出しは入力ではないことを示すために、ｆｒｒ［０－１］．Ｖ＝２に変更し、ｒｔｆ［０－１］．Ｖ＝１に変更し、ｒｔｆ［０－１］．Ｖａｌに書き込みデータを記録し、処理を終了する。

【０１１７】

一方、（４－２）該ＲＢがループの場合、ｆｒｒ［０－３１］．Ｖ＝０であれば、ｆｒｒ［０－３１］．Ｖ＝２に変更し、ｆｒｗ［０－３１］．Ｖ＝１に変更し、ｆｒｗ［０－７］．Ｖａｌに書き込みデータを記録し、処理を終了する。

【０１１８】

（５）条件コードレジスタＩＣＣ－ＲＥＡＤの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（５－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（５－２）該ＲＢがループの場合、ｉｃｒ．Ｖ＝０であれば、ｉｃｒ．Ｖ＝１に変更し、ｉｃｒ．Ｖａｌに読み出しデータを記録し、処理を終了する。

【０１１９】

（６）条件コードレジスタＩＣＣ－ＷＲＩＴＥの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（６－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（６－２）該ＲＢがループの場合、ｉｃｒ．Ｖ＝０であれば、ｉｃｒ．Ｖ＝２、ｉｃｗ．Ｖ＝１に変更し、ｉｃｗ．Ｖａｌに書き込みデータを記録し、処理を終了する。

【０１２０】

（７）浮動小数点条件コードレジスタＦＣＣ－ＲＥＡＤの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（７－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（７－２）該ＲＢがループの場合、ｆｃｒ．Ｖ＝０であれば、ｆｃｒ．Ｖ＝１に変更し、ｆｃｒ．Ｖａｌに読み出しデータを記録し、処理を終了する。

【０１２１】

（８）条件コードレジスタＩＣＣ－ＷＲＩＴＥの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして（８－１）該ＲＢが関数の場合、何もせずに処理を終了する。一方、（８－２）該ＲＢがループの場合、ｆｃｒ．Ｖ＝０であれば、ｆｃｒ．Ｖ＝２、ｆｃｗ．Ｖ＝１に変更し、ｆｃｗ．Ｖａｌに書き込みデータを記録し、処理を終了する。

【０１２２】

（９）主記憶ＲＥＡＤの場合

まず、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして、ＲＢにＷＲＩＴＥデータとして登録済である場合は、その値を使用する。一方、上記の場合ではなく、ＲＢにＲＥＡＤデータとして登録済である場合には、その値を使用する。さらに、いずれにも登録済でない場合は、キャッシュを経由して主記憶から読み込む。

【０１２３】
その後、再度ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして、（ａ）アドレスが、ＲＢに登録されているｓｐ＋６４の場合、構造体ポインタの読み出しであるので、ａｒｇ０．Ｖ＝０であれば、ａｒｇ０．Ｖ＝１に変更し、ａｒｇ０．Ｖａｌに読み出しデータを記録する。（ｂ）上記の（ａ）の場合でなく、アドレスが、ＬＩＭＩＴ以上ｓｐ＋９２未満であれば、登録不要領域であるので、何もしない。（ｃ）上記の（ｂ）の場合でない場合、ＷＲＩＴＥデータとして登録済であるかどうかを検査し、そうであれば、すでに上書きされたあとのＲＥＡＤであるので登録不要であり、何もしない。（ｄ）上記の（ｃ）でない場合、ＲＥＡＤデータとして登録済であるかどうかを検査し、そうであれば、すでに登録済であるので登録不要であり、何もしない。（ｅ）上記の（ｄ）でない場合、ＲＥＡＤデータとしての登録が必要であるので、ＲＦに主記憶ＲＥＡＤアドレスを確保し、ＲＥＡＤデータとして登録する。ＲＦに主記憶アドレスを確保できなかった場合には、登録不能であるため、そのＲＷエントリからＢＯＴＴＯＭまでに対応するＲＢエントリを全て無効化する。

【０１２４】
（１０）主記憶ＷＲＩＴＥの場合
まず、キャッシュを経由して、主記憶に書き込む。そして、ベースレジスタが１４（％ｓｐ）かつオフセットが９２以上である場合、引数の第７ワードを検出したことを記憶する。

【０１２５】
その後、ＲＷのＴＯＰからＢＯＴＴＯＭまで順にたどる。そして、（ａ）アドレスが、ＲＢに登録されているｓｐ＋６４の場合、構造体ポインタの読み出しであるので、ａｒｇ０．Ｖ＝０であれば、ａｒｇ０．Ｖ＝２に変更する。（ｂ）上記の（ａ）の場合ではなく、アドレスがＬＩＭＩＴ以上ｓｐ＋９２未満であれば、登録不要領域であるので、何もしない。（ｃ）上記の（ｂ）の場合でない場合、ＷＲＩＴＥデータとして登録済であるかどうかを検査し、そうであれば、すでにアドレスは登録済であるので、内容を新しいＷＲＩＴＥデータに更新する。（ｄ）上記の（ｃ）でない場合、ＷＲＩＴＥデータとしての登録が必要であるので、ＲＦに主記憶ＷＲＩＴＥアドレスを確保し、ＷＲＩＴＥデータとして登録する。ＲＦに主記憶アドレスを確保できなかった場合には、登録不能であるため、そのＲＷエントリからＢＯＴＴＯＭまでに対応するＲＢエントリを全て無効化する。

【０１２６】
（並列事前実行）
以上に述べた、関数やループの多重再利用では、ＲＢエントリの生存時間よりも同一パラメータが出現する間隔が長い場合や、パラメータが単調に変化し続ける場合には全く効果がないことになる。すなわち、ＲＢエントリの生存時間よりも同一パラメータが出現する間隔が長い場合には、ある関数またはループがＲＢに登録されたとしても、その登録された関数またはループに関して同一パラメータが次に出現した際には、すでにその関数またはループがＲＢエントリから消えていることになり、再利用できないことになる。また、パラメータが単調に変化し続ける場合には、該当する関数やループがＲＢに登録されていても、パラメータが異なることによって再利用できないことになる。

【０１２７】
これに対して、多重再利用を行うプロセッサとしてのＭＳＰ１Ａとは別に、命令区間の事前実行によってＲＢエントリを有効にするプロセッサとしてのＳＳＰ１Ｂを複数個設けることによって、さらなる高速化を図ることができる。

【０１２８】
並列事前実行機構を行うためのハードウェア構成は、前記した図５に示すような構成となる。同図に示すように、ＲＷ４Ａ・４Ｂ、演算器５Ａ・５Ｂ、レジスタ６Ａ・６Ｂ、キャッシュ７Ａ・７Ｂは、各プロセッサごとに独立して設けられている一方、ＲＦ／ＲＢ２、および主記憶３は全てのプロセッサが共有するようになっている。同図において、破線は、ＭＳＰ１ＡおよびＳＳＰ１ＢがＲＦ／ＲＢ２に対して入出力を登録するパスを示してい

る。

【０１２９】

ここで、並列事前実行を実現する上での課題は、（１）どのように主記憶一貫性を保つか、（２）どのように入力を予測するか、（３）どのようにＲＢエントリを入れ替えるか、（４）どの命令区間を実行するか、の４点が挙げられる。以下に、これらの課題に対する解決手法について説明する。

【０１３０】

（主記憶一貫性に関する課題の解決方法）

まず、上記の課題（１）どのように主記憶一貫性を保つかについて説明する。特に予測した入力パラメータに基づいて命令区間を実行する場合、主記憶に書き込む値がＭＳＰ１ＡとＳＳＰ１Ｂとで異なることになる。これを解決するために、図５に示すように、ＳＳＰ１Ｂは、ＲＢへの登録対象となる主記憶参照にはＲＦ／ＲＢ２、また、その他の局所的な参照にはＳＳＰ１Ｂごとに設けた局所メモリとしてのＬｏｃａｌ７Ｂを使用することとし、Ｃａｃｈｅ７Ｂおよび主記憶３への書き込みを不要としている。なお、ＭＳＰ１Ａが主記憶３に対して書き込みを行った場合には、対応するＳＳＰ１Ｂのキャッシュラインが無効化される。

【０１３１】

具体的には、ＲＢへの登録対象のうち、読み出しが先行するアドレスについては主記憶３を参照し、ＭＳＰ１Ａと同様にアドレスおよび値をＲＢへ登録する。以後、主記憶３ではなくＲＢを参照することによって、他のプロセッサからの上書きによる矛盾の発生を避けることができる。局所的な参照については、読み出しが先行するということは、変数を初期化せずに使うことに相当し、値は不定でよいことになるので、主記憶３を参照する必要はない。

【０１３２】

なお、局所メモリとしてのＬｏｃａｌ７Ｂの容量は有限であり、関数フレームの大きさがＬｏｃａｌ７Ｂの容量を超えた場合など、実行を継続できない場合は、事前実行を打ち切るようにする。また、事前実行の結果は主記憶３に書き込まれないので、事前実行結果を使って、さらに次の事前実行を行うことはできない。

【０１３３】

（入力の予測方法）

次に、上記の課題（２）どのように入力を予測するかについて説明する。事前実行に際しては、ＲＢの使用履歴に基づいて将来の入力を予測し、ＳＳＰ１Ｂへ渡す必要がある。このために、ＲＦの各エントリごとに小さなプロセッサを設け、ＭＳＰ１ＡやＳＳＰ１Ｂとは独立して入力予測値を求めるようにする。

【０１３４】

具体的には、最後に出現した引数（Ｂ）および最近出現した２組の引数の差分（Ｄ）に基づいて、ストライド予測を行う。なお、Ｂ＋Ｄに基づく命令区間の実行はＭＳＰ１Ａがすでに開始していると考える。ＳＳＰ１ＢがＮ台の場合には、用意する入力予測値は、Ｂ＋Ｄ×２からＢ＋Ｄ×（Ｎ＋１）までの範囲とする。

【０１３５】

以上のように入力予測を行えば、上記した入力パラメータが単調に変化し続けるような場合に、事前に予測しておいた結果に基づいて効果的に再利用を行うことが可能となる。

【０１３６】

（ＲＢエントリ入れ替え方法）

次に、上記の課題（３）どのようにＲＢエントリを入れ替えるかについて説明する。各ＲＦエントリが１つの命令区間に対応し、入力と出力との対応関係がＲＢに登録される際に、ＭＳＰ１ＡとＳＳＰ１ＢとがＲＢエントリをどのように使い分けるかが課題となる。命令区間は、大きく分けると、ＭＳＰ１Ａのみでも再利用の効果があるものと、配列を扱うループのようにＭＳＰ１Ａのみでは効果がないものとに分けられると考えられる。前者であれば、ＬＲＵ（ｌｅａｓｔ　ｒｅｃｅｎｔｌｙ　ｕｓｅｄ）による入れ替え、後者であ

れば、ＦＩＦＯ（Ｆｉｒｓｔ　Ｉｎ　Ｆｉｒｓｔ　Ｏｕｔ）による入れ替えが有効である。

【０１３７】

しかしながら、ある命令区間の性質がいずれであるかを動的かつ直ちに判断することは難しいので、個々のＲＦに属するＲＢエントリをＭＳＰ１Ａ用とＳＳＰ１Ｂ用とに分割し、ＭＳＰ１Ａ用のＲＢエントリをＬＲＵによって、ＳＳＰ１Ｂ用のＲＢエントリをＦＩＦＯによって入れ替えるようにする。前述したように、入力予測値はＮ組であり、ＳＳＰが登録後、ＭＳＰが直ちに利用することを想定して、ＳＳＰ用に割り当てるエントリ数はＮ×２としておく。この様子を図６に示す。

【０１３８】

（命令区間の選択）

次に、上記の課題（４）どの命令区間を実行するかについて説明する。事前実行は、前記したように、同一パラメータが出現する間隔が長い命令区間や、パラメータが単調に変化し続ける命令区間に対して効果があることが予想される。しかしながら、それぞれの命令区間の性質や実際の効果の有無は、事前に認識することはできない。そこで、ＲＦに新規に登録された命令区間については、直ちにＳＳＰ１Ｂによる数回分の事前実行を試みるようにする。そして、数回の試行の結果、ＭＳＰ１Ａによる登録頻度が高く、かつ、ＳＳＰ１Ｂが登録したエントリの再利用頻度も高いＲＦを継続してＳＳＰ１Ｂの実行対象とする。

【０１３９】

具体的には、動的に変化する登録頻度や再利用頻度を把握するために、ＲＦにおける各エントリごとに、一定期間における登録および再利用の状況をシフトレジスタに記録する。ＲＦごとに付加した小さなプロセッサが、Ｅ＝（過去の削減ステップ数）×（登録回数）×（再利用回数）を計算し、各ＳＳＰが、Ｅが最大となるＲＦを選択する。この様子を図７に示す。以下に、このことについてより詳しく説明する。

【０１４０】

事前実行するかどうかを判断する際に利用できる統計情報としては、（ａ）再利用によって削減可能なステップ数、（ｂ）ＭＳＰ１Ａが、やむをえず実行し登録した頻度、（ｃ）ＭＳＰ１Ａが、ＳＳＰ１Ｂの登録結果を再利用できた頻度、（ｄ）ＭＳＰ１Ａが、ＭＳＰ１Ａの登録結果を再利用できた頻度、が挙げられる。これらから、次にどの命令区間を事前実行すれば最大の効果が期待できるかを考える。

【０１４１】

まず、上記（ｄ）ＭＳＰ１Ａが、ＭＳＰ１Ａの登録結果を再利用できた頻度に関しては、ＳＳＰ１Ｂによる事前実行とは無関係であるので、考慮する必要はないことがわかる。

【０１４２】

上記（ａ）、（ｂ）および（ｃ）に関しては、自項目以外の項目が同じであるならば、それぞれ値が大きいほど効果を期待できることになる。また、それぞれの項目において、値が０であれば、効果を期待できないことになる。以上より、前記した式Ｅ＝（過去の削減ステップ数）×（登録回数）×（再利用回数）によって期待値を算出している。

【０１４３】

（本発明の適用例）

「ＬＩＭＩＴ」などによって大域変数領域とスタック領域とを区別できるプログラム実行環境があるとした上で、本発明に係るデータ処理装置を他の命令セットアーキテクチャにも適用するためには、スタックフレーム上の変数が、上位／下位関数のどちらの局所変数であるかを区別する手段が必要である。特に、引数を格納するレジスタが不足し、引数をスタックに格納する場合、呼ばれた関数側ではこの区別をすることができないことになる。

【０１４４】

本実施の形態で取り上げたＳＰＡＲＣプロセッサでは、引数の先頭６ワードを汎用レジスタに格納しており、６ワード以上の引数を扱う関数は出現頻度が高くないことと、引数が

スタックに溢れた時点で再利用ができなくなることの両方を利用することによって、関数／ループの再利用を実現している。ＳＰＡＲＣプロセッサ同様に、３２本以上の汎用レジスタを有する多くのＲＩＳＣプロセッサでも、同様の判断をすることによって、本発明のような関数／ループの再利用を実現することが可能である。

【０１４５】

【発明の効果】

以上のように、本発明に係るデータ処理装置は、上記主記憶手段から読み出した命令列をデコードし、該命令列に基づく演算を行う第１の演算手段と、１つ以上の命令列に関する情報を記憶する命令列記憶手段とを備え、上記第１の演算手段が、命令列をデコードした結果、該命令列が後方分岐命令に挟まれたループであるか否かを判断する第１のステップと、上記第１のステップにおいて、ループであると判断された場合に、該ループに関する情報が上記命令列記憶手段に記憶されているか否かを判断する第２のステップと、上記第２のステップにおいて、記憶されていないと判断された場合に、該ループを実行する演算を行うとともに、上記命令列記憶手段に対して、演算結果に基づいて該ループに関する入力および出力を記録する第３のステップと、上記第２のステップにおいて、記憶されていると判断された場合に、処理対象としてのループに対する入力と、上記命令列記憶手段に記憶されているループに対する入力とが一致している場合に、上記命令列記憶手段に記憶されているループに対する出力の値を、処理対象としてのループに対する出力として出力する第４のステップとを行う構成である。

【０１４６】

これにより、あるループに関する命令列が複数回行われる場合には、２回目以降では、第１の演算手段による演算処理を行うことなく、命令列記憶手段に記憶されている出力を読み出すという処理のみで対応することが可能となる。よって、処理時間の大幅な短縮を実現することができるという効果を奏する。

【０１４７】

また、本発明に係るデータ処理装置は、上記第１のステップにおいて、さらに、命令列をデコードした結果、該命令列が関数であるか否かも判断し、上記第１のステップにおいて、関数であると判断された場合に、該関数に関する情報が上記命令列記憶手段に記憶されているか否かを判断する第５のステップと、上記第５のステップにおいて、記憶されていないと判断された場合に、該関数を実行する演算を行うとともに、上記命令列記憶手段に対して、演算結果に基づいて該関数に関する入力および出力を記録する第６のステップと、上記第５のステップにおいて、記憶されていると判断された場合に、処理対象としての関数に対する入力と、上記命令列記憶手段に記憶されている関数に対する入力とが一致している場合に、上記命令列記憶手段に記憶されている関数に対する出力の値を、処理対象としての関数に対する出力として出力する第７のステップとを行う構成としてもよい。

【０１４８】

これにより、上記の構成による効果に加えて、命令列が関数である場合にも、上記のループと同様に値再利用を実現することができるので、処理時間の大幅な短縮を実現することができるという効果を奏する。

【０１４９】

また、本発明に係るデータ処理装置は、上記第３のステップにおいて、上記命令列記憶手段に対して、主記憶手段における該ループの開始アドレスおよび終了アドレスと、全ての入出力に関する主記憶アドレスとを記録するとともに、上記第４のステップにおいて、処理対象としてのループの開始アドレス、ならびに、入力に関する主記憶アドレスが、命令列記憶手段に記憶されているものと完全に一致した場合に、上記命令列記憶手段に記憶されているループに対する出力の値を、処理対象としてのループに対する出力として出力する構成としてもよい。

【０１５０】

これにより、上記の構成による効果に加えて、的確にループを特定することが可能となり、ループに関する再利用を実現することが可能となるという効果を奏する。

【０１５１】
また、本発明に係るデータ処理装置は、上記第６のステップにおいて、上記命令列記憶手段に対して、主記憶手段における該関数の先頭アドレスと、入出力に関する主記憶アドレスとを記録するとともに、上記第７のステップにおいて、処理対象としての関数の先頭アドレス、ならびに、入力に関する主記憶アドレスが、命令列記憶手段に記憶されているものと完全に一致した場合に、上記命令列記憶手段に記憶されている関数に対する出力の値を、処理対象としての関数に対する出力として出力する構成としてもよい。
【０１５２】
これにより、上記の構成による効果に加えて、的確に関数を特定することが可能となり、関数に関する再利用を実現することが可能となるという効果を奏する。
【０１５３】
また、本発明に係るデータ処理装置は、上記主記憶手段から読み出された入力を一時的に格納し、上記第１の演算手段にわたすとともに、上記第１の演算手段からの出力を一時的に格納し、上記主記憶手段にわたすレジスタをさらに備えるとともに、上記命令列記憶手段に、ループの入力および出力が格納される全てのレジスタの情報が記録される構成としてもよい。
【０１５４】
これにより、上記の構成による効果に加えて、第１の演算手段は、レジスタの値に基づいて、再利用が可能か否かの判断を行うことができるので、この判断を的確かつ迅速に行うことが可能となるという効果を奏する。
【０１５５】
また、本発明に係るデータ処理装置は、特定の上位の命令区間が、その内部に１つ以上の下位の命令区間を含んでいる場合に、下位の命令区間の実行中に、該下位の命令区間が行った入出力が上位の命令区間の局所変数である場合には、該下位の命令区間の入出力として上記命令列記憶手段に登録し、該下位の命令区間が行った入出力が上位の命令区間の局所変数でない場合には、該下位の命令区間および上位の命令区間の入出力として上記命令列記憶手段に登録する構成としてもよい。
【０１５６】
これにより、上記の構成による効果に加えて、例えば命令列記憶手段に上位の命令区間に関する情報が登録されている場合に、該命令区間が出現した場合に、その入力を参照することによって、下位の命令区間を考慮することなく、的確に該命令区間の再利用を実現することが可能となるという効果を奏する。
【０１５７】
また、本発明に係るデータ処理装置は、上記命令列記憶手段に対して、上位の命令区間および１つ以上の下位の命令区間が登録されている場合に、登録されている命令区間の包含関係の順番をスタック構造で記憶する再利用記憶手段をさらに備えている構成としてもよい。
【０１５８】
これにより、上記の構成による効果に加えて、命令列記憶手段に多重入出力構造の命令区間が複数登録されていても、再利用記憶手段を参照することによって、各命令区間の包含関係を認識することが可能となるという効果を奏する。
【０１５９】
また、本発明に係るデータ処理装置は、上記再利用記憶手段に登録可能な命令区間の多重度を超えて、命令区間を検出した場合、その時点で最も上位の命令区間の登録を中止する構成としてもよい。
【０１６０】
これにより、上記の構成による効果に加えて、再利用可能な多重入出力構造の登録を最低限確保することが可能となるという効果を奏する。
【０１６１】
また、本発明に係るデータ処理装置は、少なくとも１つの第２の演算手段をさらに備え、

上記第２の演算手段が、上記第１の演算手段によって処理が行われている命令列に関して、今後入力が予想される予測入力値に基づいて該命令列の演算を行い、その結果を上記命令列記憶手段に対して登録する構成としてもよい。

【０１６２】

これにより、上記の構成による効果に加えて、次に、同じ命令列が出現し、予測入力値と同じ入力が行われた場合には、命令列記憶手段に記憶されている値を再利用することが可能となるという効果を奏する。

【０１６３】

また、本発明に係るデータ処理装置は、上記第１の演算手段に対して、主記憶手段に対して読み出しおよび書き込みが可能なキャッシュメモリが設けられているとともに、上記第２の演算手段に対して、主記憶手段に対して読み出しのみ可能なキャッシュメモリが設けられており、上記第２の演算手段が、命令列記憶手段に対する入出力記録対象となる主記憶参照には命令列記憶手段に記録されている入出力記録そのものを用い、その他の局所的な参照には、上記第２の演算手段に設けられたローカルメモリを用いるとともに、上記第１の演算手段が主記憶手段へ書き込みを行う場合には、対応する第２の演算手段のキャッシュラインが無効化される構成としてもよい。

【０１６４】

これにより、上記の構成による効果に加えて、第１の演算手段と、第２の演算手段とで、主記憶手段に対して異なる値を書き込むことによる不具合を防止することができるという効果を奏する。

【０１６５】

また、第１の演算手段によって主記憶手段が書き換えられた場合にも、第２の演算手段が、新たに主記憶手段から値を読み出すことによって、主記憶一貫性を保つことができるという効果を奏する。

【０１６６】

また、本発明に係るデータ処理装置は、上記命令列記憶手段において、上記第１の演算手段による書き込み範囲と、上記第２の演算手段による書き込み範囲とを分割するとともに、上記第１の演算手段による書き込み範囲では、ＬＲＵ（ｌｅａｓｔ　ｒｅｃｅｎｔｌｙ　ｕｓｅｄ）によってエントリの入れ替えが行われるようにし、上記第２の演算手段による書き込み範囲では、ＦＩＦＯ（Ｆｉｒｓｔ　Ｉｎ　Ｆｉｒｓｔ　Ｏｕｔ）によってエントリの入れ替えが行われるようにする構成としてもよい。

【０１６７】

これにより、上記の構成による効果に加えて、エントリの入れ替えは、第１の演算手段による書き込み範囲ではＬＲＵ、第２の演算手段による書き込み範囲ではＦＩＦＯとしているので、より効率的に再利用を行うことが可能となるという効果を奏する。

【０１６８】

また、本発明に係るデータ処理装置は、命令列記憶手段における各エントリごとに、一定期間における登録および再利用の状況が記録され、Ｅ＝（過去の削減ステップ数）×（登録回数）×（再利用回数）が算出されるとともに、上記第２の演算手段が、上記Ｅが最大となるエントリに関して事前実行を行う構成としてもよい。

【０１６９】

これにより、上記の構成による効果に加えて、第１の演算手段による登録頻度が高く、かつ、第２の演算手段が登録したエントリの再利用頻度も高い命令列を継続して第２の演算手段の実行対象とすることができる。よって、第２の演算手段による事前実行の効果を最大限高めることが可能となるという効果を奏する。

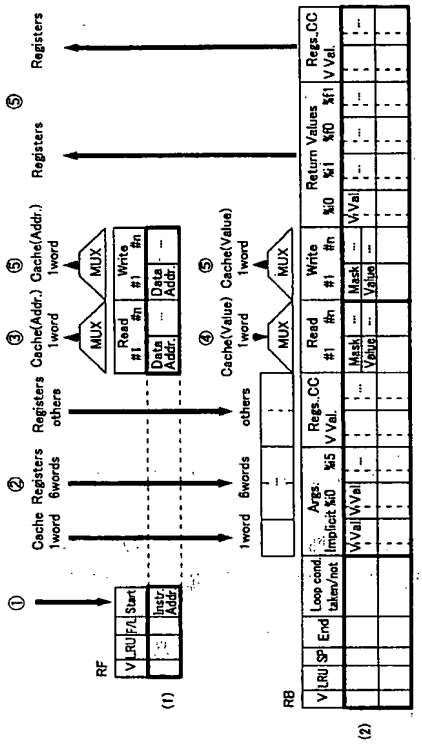
【図面の簡単な説明】

【図１】本発明の一実施形態に係るデータ処理装置が備えるＲＦ／ＲＢによって実現される再利用表を示す図である。

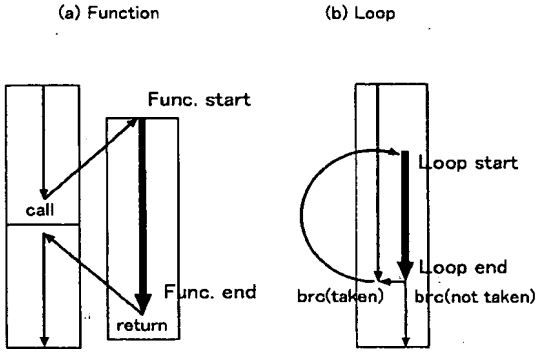
【図２】同図（ａ）は、関数が呼び出されて実行される様子を概念的に示す図であり、同図（ｂ）は、ループ処理が実行される様子を概念的に示す図である。

【図３】関数の入れ子構造において、内側の構造のレジスタ入出力が、外側の構造のレジスタ入出力となる影響範囲を示す図である。
【図４】ＲＷと、ＲＦ・ＲＢとの関係を示す図である。
【図５】上記データ処理装置の概略構成を示すブロック図である。
【図６】ＲＢエントリを分割する状態を示す図である。
【図７】ＲＦにおける各エントリごとに、一定期間における登録および再利用の状況をシフトレジスタに記録する様子を示す図である。
【図８】関数およびループが入れ子構造となっている状態の一例を示す図である。
【図９】命令がデコードされた結果、関数呼び出し命令である場合の処理の流れを示すフローチャートである。
【図１０】命令がデコードされた結果、関数復帰命令である場合の処理の流れを示すフローチャートである。
【図１１】命令がデコードされた結果、後方分岐成立である場合の処理の流れを示すフローチャートである。
【図１２】命令がデコードされた結果、後方分岐不成立である場合の処理の流れを示すフローチャートである。
【図１３】同図（ａ）は、関数Ａが関数Ｂを呼び出す構造を概念的に示す概念図であり、同図（ｂ）は、同図（ａ）に示すプログラム構造を実行する際の主記憶におけるメモリマップを示す図である。
【図１４】関数Ａが関数Ｂを呼び出す場合の、メモリマップにおける引数およびフレームの概要を示す図である。
【図１５】１つの関数を再利用するための従来の再利用表を示す図である。
【符号の説明】
１Ａ　ＭＳＰ
１Ｂ　ＳＳＰ
２　ＲＦ／ＲＢ（命令列記憶手段）
３　主記憶（主記憶手段）
４Ａ・４Ｂ　ＲＷ（再利用記憶手段）
５Ａ・５Ｂ　演算器（第１・第２の演算手段）
６Ａ・６Ｂ　レジスタ
７Ａ・７Ｂ　Ｃａｃｈｅ

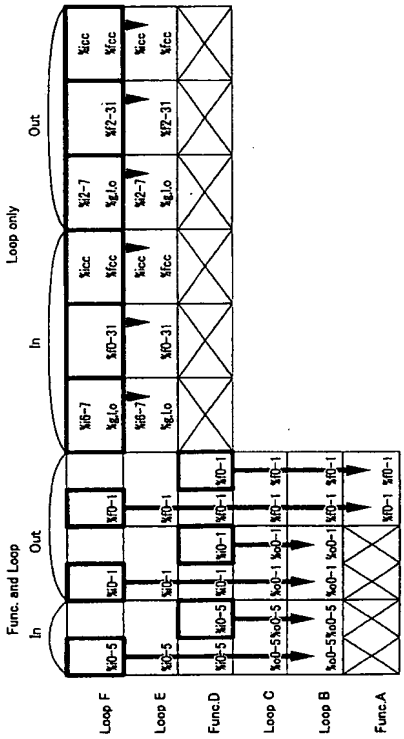
【図１】


Registers
Cache(Addr.) 1word
Cache(Value) 1word
Registers others
Cache Registers 1word 6words
Args. Implicit %i0 %i5
Regs.CC
Loop cond. taken/not
Return Values %i0 %i1 %f0 %f1
RF
RB
(1)
(2)

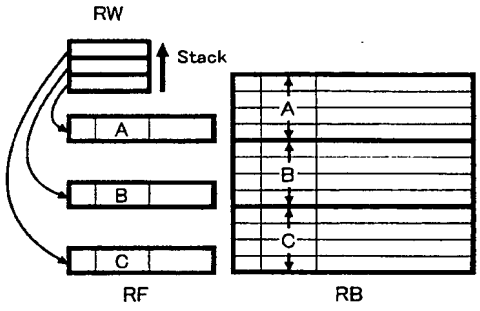
【図２】


(a) Function
(b) Loop
Func. start
call
Func. end
return
Loop start
Loop end
brc(taken)
brc(not taken)

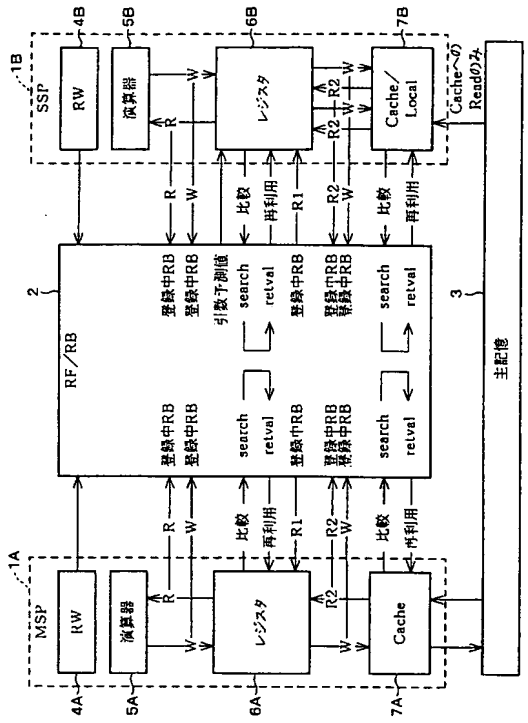
【図３】


Out
Loop only
In
Func. and Loop
Out
In
Loop F
Loop E
Func.D
Loop C
Loop B
Func.A

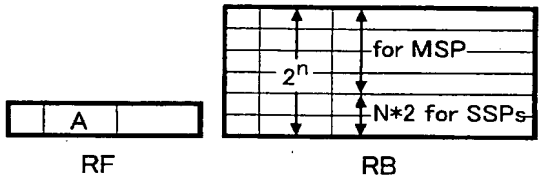
【図４】


RW
Stack
A
B
C
RF
RB

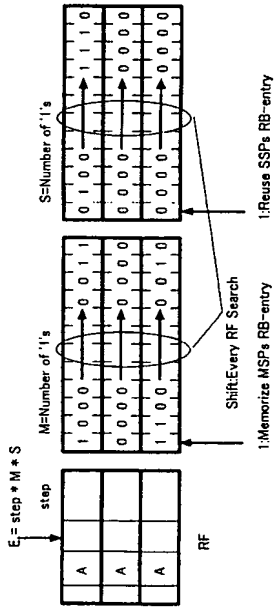
【図５】


1B
SSP
4B
RW
5B
演算器
6B
レジスタ
7B
Cache/
Local
Cache～のReadのみ
2
RF／RB
3
主記憶
1A
MSP
4A
5A
6A
7A
Cache
登録中RB
引数予測値
search
retval
比較
再利用
R
W
R1
R2

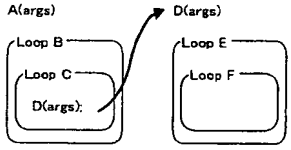
【図６】


A
RF
2n
for MSP
N*2 for SSPs
RB

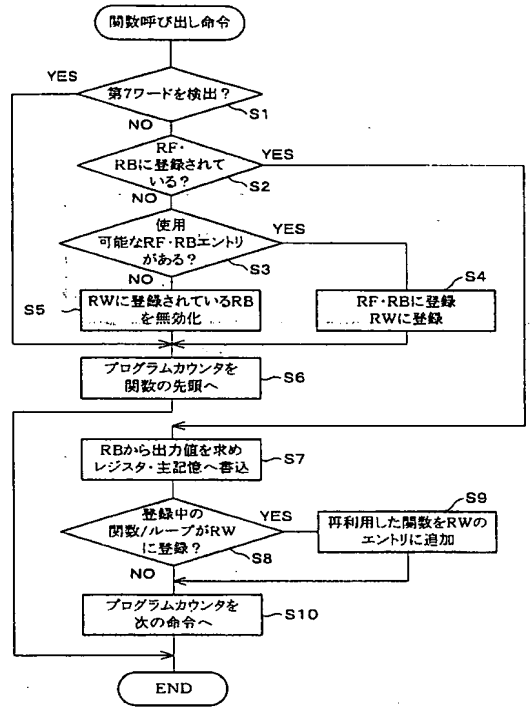
【図７】


S=Number of '1's
1:Reuse SSPs RB-entry
M=Number of '1's
Shift:Every RF Search
1:Memorize MSPs RB-entry
E = step * M * S
step
A
RF

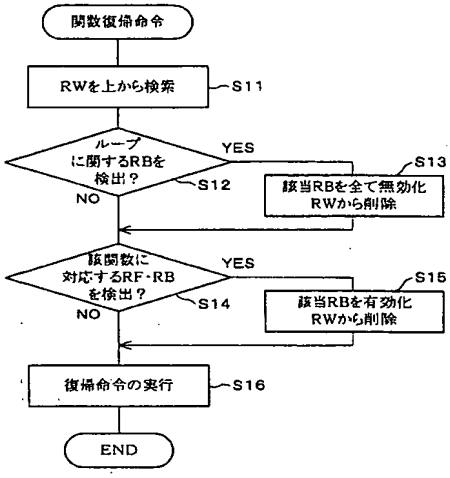
【図８】


A(args)
D(args)
Loop B
Loop C
D(args);
Loop E
Loop F

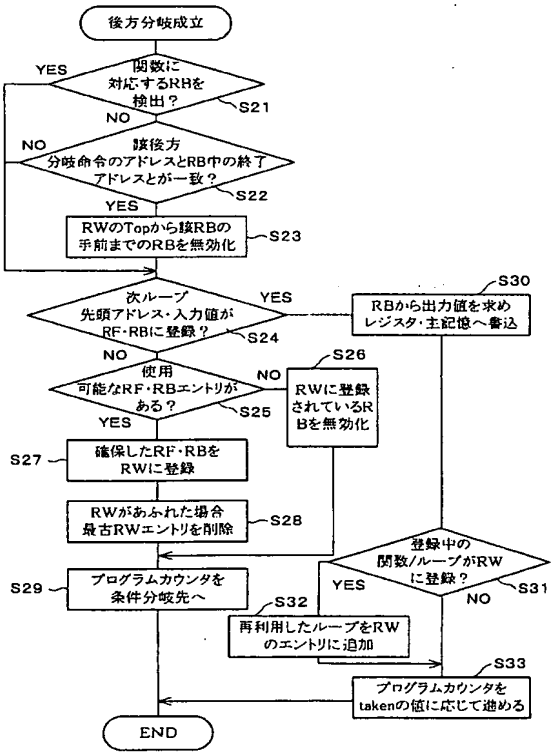
【図９】



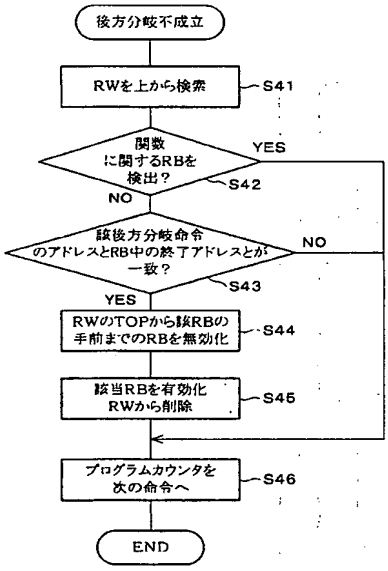
【図１０】



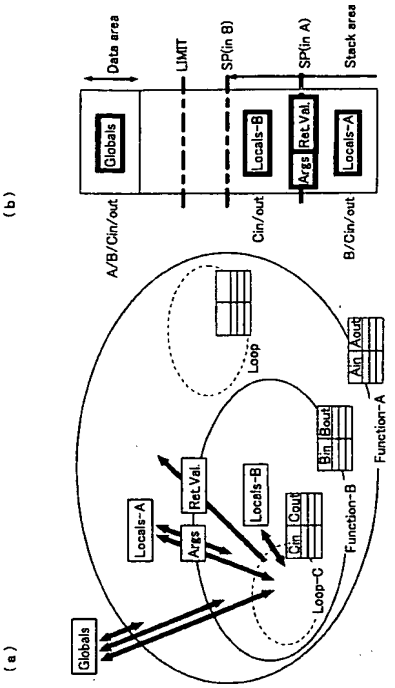
【図１１】



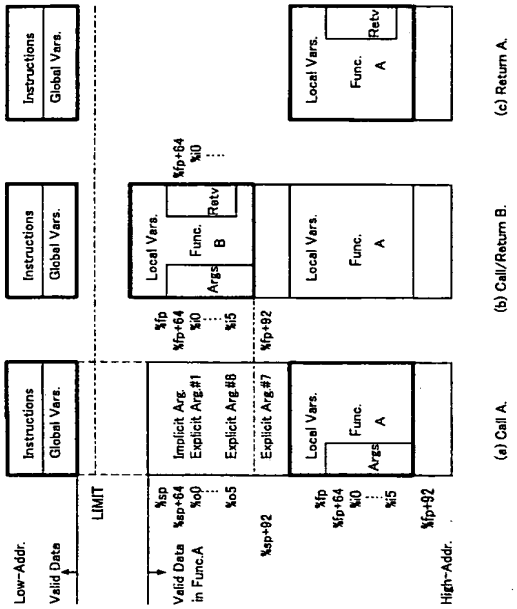
【図１２】



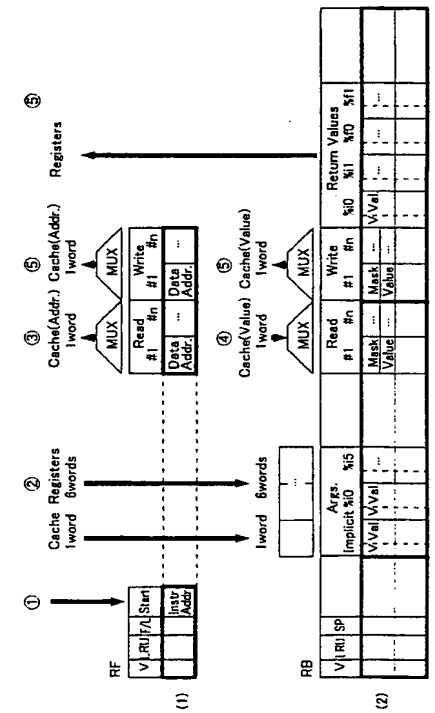
【図１３】


(a)
Globals
Loop-C
Function-B
Function-A
Loop
Locals-A
Args
Ret.Val.
Locals-B
Cin
Cout
Bin
Bout
Ain
Aout
(b)
Globals
Data area
LIMIT
SP(in B)
SP(in A)
Stack area
A/B/Cin/out
Cin/out
B/Cin/out
Locals-B
Args
Ret.Val.
Locals-A

【図１４】


Low-Addr.
Valid Data
LIMIT
High-Addr.
Instructions
Global Vars.
Local Vars.
Func. A
Args
Retv
Func. B
Implicit Arg.
Explicit Arg.#1
Explicit Arg.#8
Explicit Arg.#7
Valid Data in Func.A
%sp
%sp+64
%o0
%o5
%sp+92
%fp
%fp+64
%i0
%i5
%fp+92
(a) Call A.
(b) Call/Return B.
(c) Return A.

【図１５】


RF
RB
V
RU
SP
Start
Instr. Addr.
Args.
Implicit %i0
%i5
Return Values
%i0
%i1
%f0
%f1
Cache Registers 6words
Iword
6words
Cache(Addr.) Iword
Cache(Value) Iword
Read #1 #n
Write #1 #n
Data Addr.
Mask
Value
MUX
Registers
(1)
(2)